PanaXSeries

The One to Watch for Constant Innovation-Making the Future Come Alive

MICROCOMPUTER MN102L

MN102L Series Instruction Manual

Pub.No.12250-030E

Panasonic

Request for your special attention and precautions in using the technical informaition and semiconductors described in this book

(1) An export permit needs to be obtained from the competent authorities of the Japanese Government if any of the products or technologies described in this book and controlled under the "Foreign Exchange and Foreign Trade Law" is to be exported or taken out of Japan.

(2) The contents of this book are subject to change without notice in matters of improved function.When finalizing your design, therefore, ask for the most up-to-date version in advance in order to check for any changes.

(3) We are not liable for any damage arising out of the use of the contents of this book, or for any infringement of patents or any other rights owned by a third party.

(4) No part of this book may be reprinted or reproduced by any means without written permission from our company.

If you have any inquiries or questions about this book or our semiconductors, please contact one of our sales offices listed at the back of this book or Matsushita Electronics Corporation's Sales Department.

About This Manual

This manual describes in detail the instruction set for the MN102L Series. Chapter 1 explains the functions, basic format and instruction execution times of the instruction set. Chapter 2 describes the operation of each instruction and the flags changed by each. Chapter 3 provides cautions and warnings for the use of instructions. The appendix contains a summary of the instruction set and an instruction map.

■Searching for information

This manual has four types of indexing to speed up searches for necessary information.

(1) To find a start of chapter, refer to the index at the start of the manual.
(2) To find a title, refer to the table of contents at the start of the manual.
(3) The chapter title for each page is shown at the top of the right-hand page, and the section title is shown at the bottom. You can get a brief idea of the contents while flipping through the pages.
(4) To find an instruction, refer to the index at the end of the manual. Also, an instruction index can be found on the right-hand pages, so you can search for an instruction while flipping through the pages.

■Related manuals

Matsushita provides the following manuals related to the product covered in this manual.

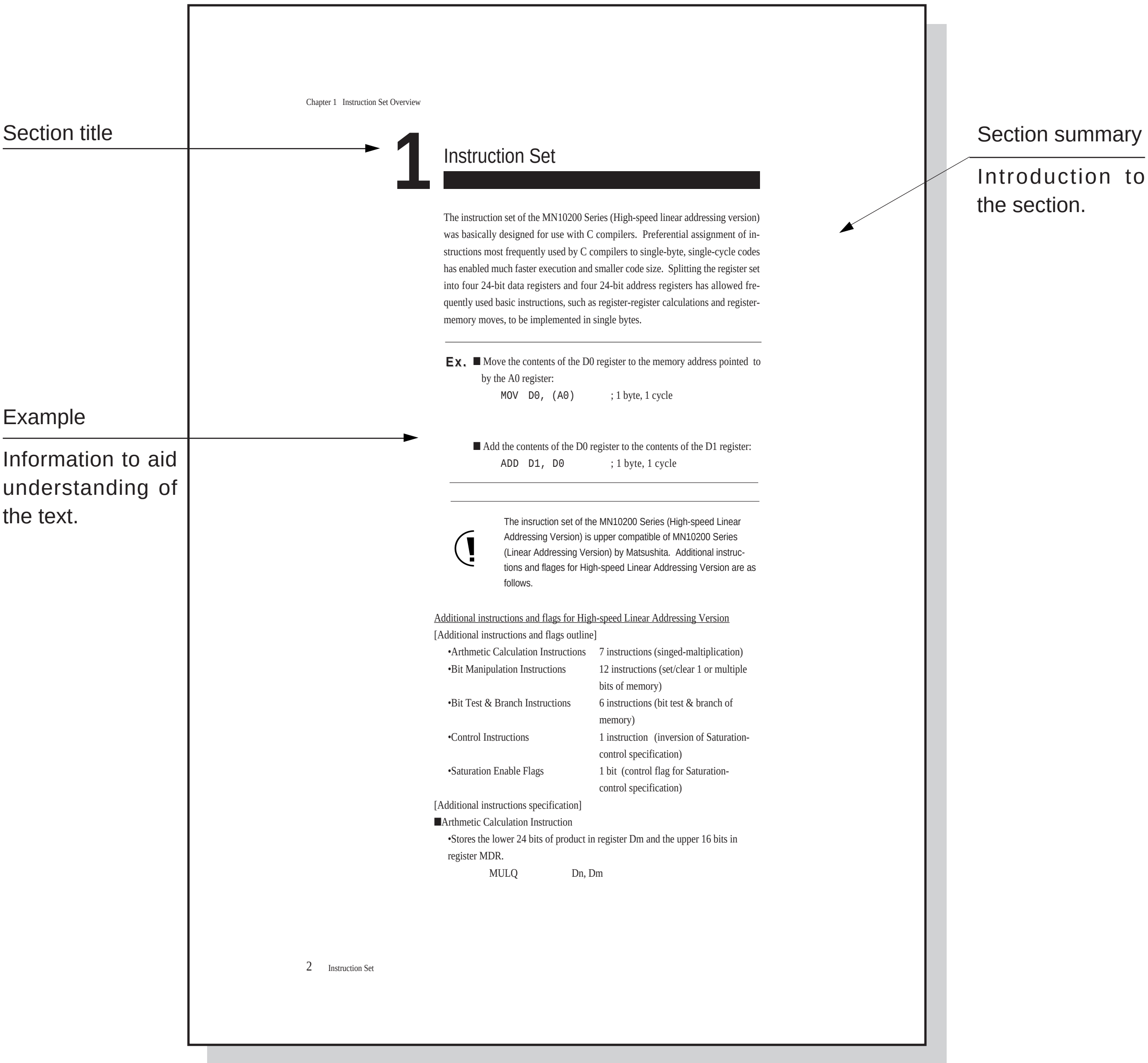
- "MN102L Series LSI User's Manual"
  <Describes the device hardware>
- "MN102L Series Cross-assembler User's Manual"
  <Describes the assembler syntax and notation>
- "MN102L Series C Compiler User's Manual: Usage Guide"
  <Describes the installation, the commands, and options of the C Compiler>
- "MN102L Series C Compiler User's Manual: Language Description"
  <Describes the syntax of the C Compiler>
- "MN102L Series C Compiler User's Manual: Library Reference"
  <Describes the standard library of the C Compiler>
- "MN102L Series C Source Code Debugger User's Manual"
  <Describes the use of the C source code debugger>
  (Note: For C Source Code Debugger for Windows®, this manual is not necessary.)
- "MN102H Series C Source Code Debugger for Windows® User's Manual"
  <Describes the use of the C source code debugger for Windows®>
- "MN102H Series Installation Manual"
  <Describes the installation of the C compiler, cross-assembler and C source code debugger and the procedure for bringing up the in-circuit emulator>

■Contacting Matsushita

Please send any comments or questions regarding the contents of this manual to your nearest Semiconductor Design Center (refer to the list of addresses at the back of the manual).

■Layout of the manual

Chapter 1 broadly consists of section titles and summaries, text, examples, and important notes. Chapter 2 consists of instruction commands, operation descriptions, and important notes. The following diagrams show the layout and meaning of each page.



Chapter 1 Instruction Set Overview

1 Instruction Set

The instruction set of the MN10200 Series (High-speed linear addressing version) was basically designed for use with C compilers. Preferential assignment of instructions most frequently used by C compilers to single-byte, single-cycle codes has enabled much faster execution and smaller code size. Splitting the register set into four 24-bit data registers and four 24-bit address registers has allowed frequently used basic instructions, such as register-register calculations and register-memory moves, to be implemented in single bytes.

Ex. ■ Move the contents of the D0 register to the memory address pointed to by the A0 register:

```
MOV  D0, (A0)        ; 1 byte, 1 cycle
```

■ Add the contents of the D0 register to the contents of the D1 register:

```
ADD  D1, D0          ; 1 byte, 1 cycle
```

The insruction set of the MN10200 Series (High-speed Linear Addressing Version) is upper compatible of MN10200 Series (Linear Addressing Version) by Matsushita. Additional instructions and flages for High-speed Linear Addressing Version are as follows.

Additional instructions and flags for High-speed Linear Addressing Version

[Additional instructions and flags outline]

•Arthmetic Calculation Instructions	7 instructions (singed-maltiplication)
•Bit Manipulation Instructions	12 instructions (set/clear 1 or multiple bits of memory)
•Bit Test & Branch Instructions	6 instructions (bit test & branch of memory)
•Control Instructions	1 instruction (inversion of Saturation-control specification)
•Saturation Enable Flags	1 bit (control flag for Saturation-control specification)

[Additional instructions specification]

■Arthmetic Calculation Instruction

•Stores the lower 24 bits of product in register Dm and the upper 16 bits in register MDR.

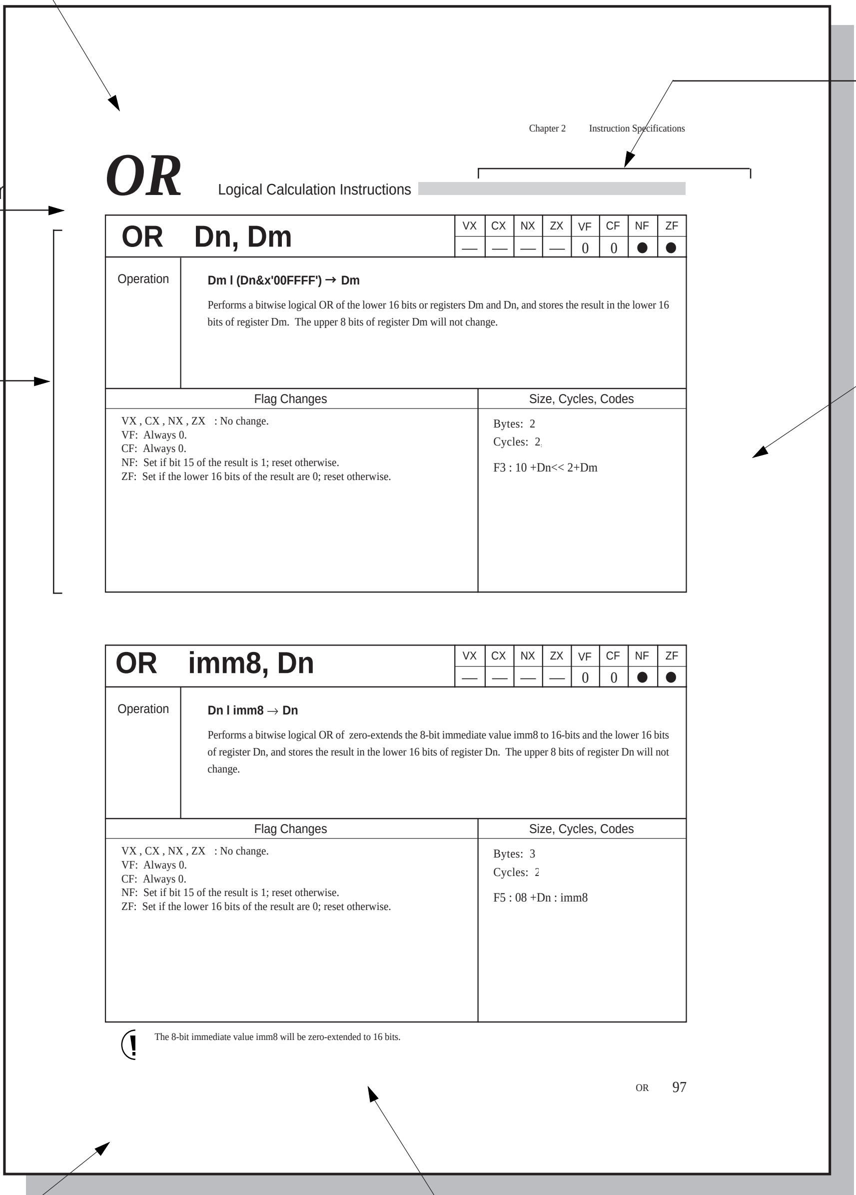
MULQ Dn, Dm

2 Instruction Set

Instruction function and type

Instruction form

Operation description



Chapter 2 Instruction Specifications

OR Logical Calculation Instructions

OR Dn, Dm	VX	CX	NX	ZX	VF	CF	NF	ZF
	—	—	—	—	0	0	●	●

Operation

Dm I (Dn&x'00FFFF') → Dm

Performs a bitwise logical OR of the lower 16 bits or registers Dm and Dn, and stores the result in the lower 16 bits of register Dm. The upper 8 bits of register Dm will not change.

Flag Changes	Size, Cycles, Codes
VX , CX , NX , ZX : No change. VF: Always 0. CF: Always 0. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 2 Cycles: 2 F3 : 10 +Dn<< 2+Dm

OR imm8, Dn	VX	CX	NX	ZX	VF	CF	NF	ZF
	—	—	—	—	0	0	●	●

Operation

Dn I imm8 → Dn

Performs a bitwise logical OR of zero-extends the 8-bit immediate value imm8 to 16-bits and the lower 16 bits of register Dn, and stores the result in the lower 16 bits of register Dn. The upper 8 bits of register Dn will not change.

Flag Changes	Size, Cycles, Codes
VX , CX , NX , ZX : No change. VF: Always 0. CF: Always 0. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 3 Cycles: 2 F5 : 08 +Dn : imm8

! The 8-bit immediate value imm8 will be zero-extended to 16 bits.

OR 97

VX/CX/NX/ZX/VF/CF/NF/ZF changes

May change	●
Will not change	—
Undefined	?
Always 0	0
Always 1	1

Code size, cycles, instruction code

Indicates the code size, number of cycles (minimum), and instruction code when the instruction format is used. The bytes of the instruction code are delimited by colons (:). A "<<2" will indicate a 2-bit shift, and a register name will be converted to the corresponding register number in code.

Footer

Indicates the instruction.

Warning

Read thoroughly to ensure correct operation of programs.

Table of Contents

Chapter 1 Instruction Set Overview

Chapter 2 Instruction Specifications

Arithmetic Calculation Instructions

Logical Calculation Instructions

Bit Manipulation Instructions

Branch Instructions

Chapter 3 Use Of Instructions

Appendix

Index

Instruction Set Overview

1

1 Instruction Set

The instruction set of the MN102L Series was basically designed for use with C compilers. Preferential assignment of instructions most frequently used by C compilers to single-byte, single-cycle codes has enabled much faster execution and smaller code size. Splitting the register set into four 24-bit data registers and four 24-bit address registers has allowed frequently used basic instructions, such as register-register calculations and register-memory moves, to be implemented in single bytes.

Ex. ■ Move the contents of the D0 register to the memory address pointed to by the A0 register:

```
MOV     D0, (A0)        ; 1 byte, 1 cycle
```

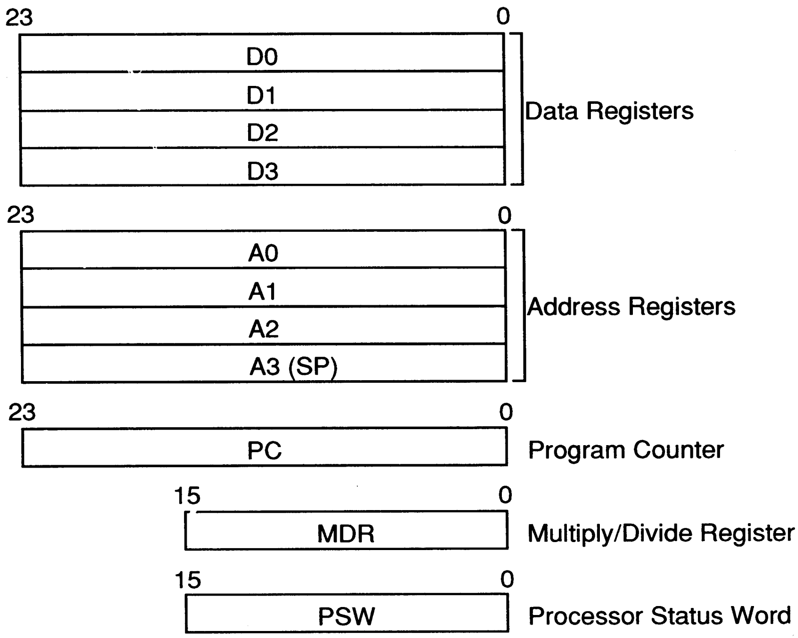
■ Add the contents of the D0 register to the contents of the D1 register:

```
ADD     D1, D0          ; 1 byte, 1 cycle
```

---

2 Register Set

The register set is divided by function into data registers for calculations and address registers ror pointers. This greatly contributes to improvements in instruction code size compression and pipeline processing parallelism.When Matsushita's C compiler is used, this configurations shrinks code size to the maximum extent possible.Address registers are 24 bits, enabling use of a contiguous memory space up to 16 Mbytes.



2-1 Data Registers

D3～D0 : Data Register (4 x 24 bits)

Data registers can generally be used for any calculation.

Calculations are performed in 24 bits, but a flag change can cause calculations with both 24 bits and the lower 16 bits. The data size will be changed by memory data move instructions and specialized size extension instructions. When 8-bit data is loaded, it will be sign-extended or zero-extended to 24 bits and moved to a register. When stored, the lower 8 bits of the register will be moved to memory. When 16-bit data is loaded, it will be sign-extended to 24 bits and moved to a register. When stored, the lower 16 bits of the register will be moved to memory. Finally, 24-bit data will be moved to and from memory as is.

2-2 Address Registers

A3～A0 : Address Register (4 x 24 bits)

Address registers are used as address pointers. They are supported only by instructions for address calculations (add, subtract, compare). Calculations are performed in 24 bits, but a flag change can cause calculations with both 24 bits and the lower 16 bits. A3 is assigned as the stack pointer. Moves to and from memory are always as 24 bits.

2-3 Program Counter

PC : Program Counter (1 x 24 bits)

The program counter is a 24-bit counter that indicates the address of the instruction currently executing.

2-4 Multiply/Divide Register

MDR : Multiply/Divide Register (1 x 16 bits)

The multiply/divide register is provided for multiply and divide instructions. For multiply instructions, it will store the upper 16 bits of the 32-bit product. For divide instructions, it will store the upper 16 bits of the dividend before execution and the 16-bit remainder of the result after execution.

2-5 Processor Status Word

PSW : Processor Status Word (1 x 16 bits)

The processor status word is a register that indicates the CPU state. It contains flags for calculation results and interrupt mask levels.

15							8	7							0
—	—	S1	S0	IE	IM2	IM1	IM0	VX	CX	NX	ZX	VF	CF	NF	ZF

ZF : Zero Flag

The zero flag is set if the lower 16 bits of a calculation result are all 0; otherwise the flag is cleared.

NF : Negative Flag

The negative flag is set if bit 15 of a calculation result is '1'; if '0' then the flag is cleared.

CF : Carry Flag

The carry flag is set if a calculation generates a carry from or a borrow to bit 15; if not generated then the flag is cleared.

Ex. The CF is set if the result of a calculation cannot be expressed by a 16-bit unsigned value in the range x'0000' to x'FFFF'.

```
MOV    x'7FFF', D0
ADD    x'89AB', D0    ;CF=1

x'7FFF' +x'89AB' =x'109AA'
                  (Carry from)
```

VF : Overflow Flag

The overflow flag is set if a calculation resulting in a 16-bit signed value generates an overflow; if not generated then the flag is cleared.

Ex. The VF is set if the result of a calculation cannot be expressed by a 16-bit signed value in the range x'8000' (the most negative value) to x'7FFF' (the most positive value).

```
MOV    x'7654', D0
ADD    x'4321', D0    ;VF=1

x'7654'    +x'4321'    =x'B975'
(positive)  (positive)  (negative)
```

ZX : Extended Zero Flag
The extended zero flag is set if 24 bits of a calculation result are all '0'; otherwise the flag is cleared.

NX : Extended Negative Flag
The extended negative flag is set if the most significant bit of a calculation result is '1'; if '0' then the flag is cleared.

CX : Extended Carry Flag
The extended carry flag is set if a calculation generates a carry from or a borrow to the most significant bit; if not generated then the flag is cleared.

VX : Extended Overflow Flag
The extended overflow flag is set if a calculation resulting in a 24-bit signed value generates an overflow; if not generated then the flag is cleared.

IM2~IM0 : Interrupt Mask
The interrupt mask bits indicate the CPU interrupt mask level. These three bits can be set from level '0' (000) to level '7' (111). Level 0 is the highest interrupt level.

IE : Interrupt Enable
The interrupt enable bit enables reception of interrupts when set to '1'. Note that non-maskable interrupts will always be received regardless of the state of IE.

S1, S0 : Software Bit
The software bits are used by the operating system as software control bits.

(!) The value of PSW will be x'0000' when started after a reset.

---

3 Instruction Functions

The instructions of this series can be divided by function into five categories.

- data move instructions
- arithmetic calculation instructions
- logical calculation instructions
- bit manipulation instructions
- branch instructions

3-1 Data Move Instructions

The data move instructions are as follows.

MOV	Move Data/Move Pointer Data
MOVX	Move Pointer Data
MOVB	Move Byte
MOVBU	Move Byte Unsigned
EXT	Extention
EXTX	Extension Word Data to Pointer Data
EXTXU	Extension Word Data Unsigned to Pointer Data
EXTXB	Extension Byte Data to Pointer Data
EXTXBU	Extension Byte Data Unsigned to Pointer Data

MOV instructions move 16-bit data in data registers (Dn) and 24-bit pointer data in address registers (An).
MOVX instructions move 24-bit pointer data in data registers (Dn).
MOVB and MOVBU instructions move byte data between memory and data registers. The MOVB instruction sign-extends 8-bit data in memory to 24 bits when moving from memory to a register, and moves the lower 8 bits of data in a register when moving from the register to memory. The MOVBU instruction zero-extends 8-bit data in memory to 24 bits when moving.
EXT instructions sign-extend 16-bit data in data registers (Dn) to 32 bits. The upper 16 bits that are sign-part are moved to MDR, where they will be used for 32-bit calculations.
EXTX instructions sign-extend 16-bit data in data registers (Dn) to 24 bits.
EXTXU instructions zero-extend 16-bit data in data registers (Dn) to 24 bits.
EXTXB instructions sign-extend 8-bit data in data registers (Dn) to 24 bits.
EXTXBU instructions zero-extend 8-bit data in data registers (Dn) to 24 bits.

3 - 2 Arithmetic Calculation Instructions

The arithmetic calculation instructions are as follows.

ADD	Addition
ADDC	Addition with Carry
ADDNF	Addition with Non Flag Change
SUB	Subtract
SUBC	Subtract with Borrow
MUL	Multiply
MULU	Multiply Unsigned
DIVU	Divide Unsigned
CMP	Compare

Arithmetic calculations include addition, subtraction, multiplication, division, and comparison. Multiplication can be signed or unsigned.

3 - 3 Logical Calculation Instructions

The logical calculation instructions are as follows.

AND	And
OR	Or
XOR	Exclusive Or
NOT	Complement
ASR	Arithmetical Shift Right
LSR	Logical Shift Right
ROR	Rotate Right with Carry
ROL	Rotate Left with Carry

All logical calculation instructions are performed on data registers. The AND and OR instructions can also operate with immediate data on the processor status word (PSW). For a left shift, use ADD Dn, Dn (where Dn is the same for both operands).

3 - 4 Bit Manipulation Instructions

The bit manipulation instructions are as follows.

BTST	Bit Test
BSET	Bit Test and Set
BCLR	Bit Test and Clear

BTST instructions test data register contents with immediate data. The result of a logical OR of the data register and immediate value will be reflected in the flags. BSET and BCLR instructions test memory while the bus is locked and interrupts disabled, reflect the result in the flags, and then set/clear the specified bit. BSET and BCLR instructions access memory as 8 bytes.

3 - 5 Branch Instructions

The branch instructions are as follows.

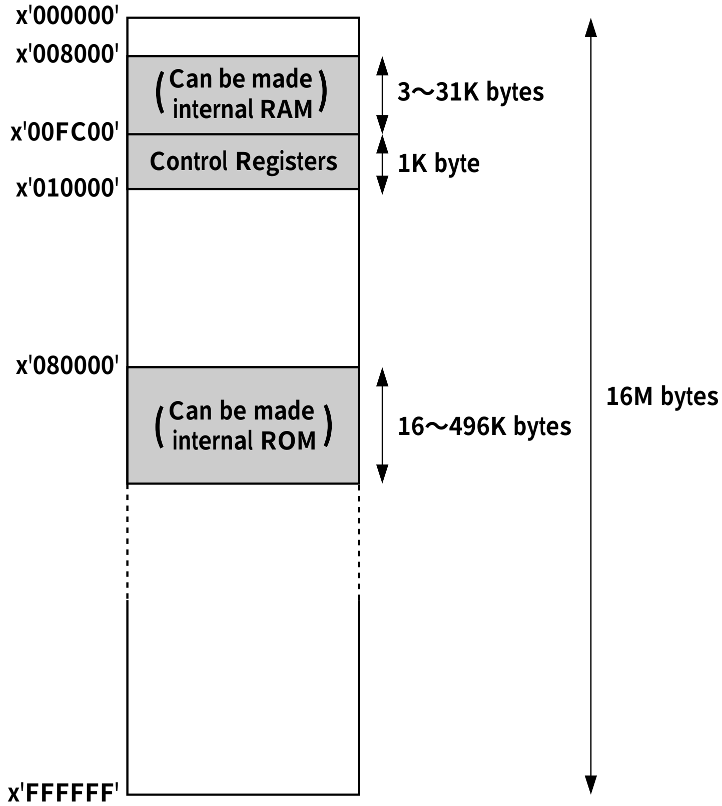
BEQ	Branch Equal
BNE	Branch Not Equal
BGT	Branch Greater Than
BGE	Branch Greater or Equal
BLE	Branch Less or Equal
BLT	Branch Less Than
BHI	Branch Higher
BCC	Branch Carry Flag Clear
BLS	Branch Low or Same
BCS	Branch Carry Flag Set
BVC	Branch Overflow Flag Clear
BVS	Branch Overflow Flag Set
BNC	Branch Negative Flag Clear
BNS	Branch Negative Flag Set
BRA	Branch Always
BEQX	Branch Equal by Extended Flags
BNEX	Branch Not Equal by Extended Flags
BGTX	Branch Greater Than by Extended Flags
BGEX	Branch Greater or Equal by Extended Flags
BLEX	Branch Less or Equal by Extended Flags
BLTX	Branch Less Than by Extended Flags
BHIX	Branch Higher by Extended Flags
BCCX	Branch Extended Carry Flag Clear
BLSX	Branch Low or Same by Extended Flags
BCSX	Branch Extended Carry Flag Set
BVCX	Branch Extended Overflow Flag Clear
BVSX	Branch Extended Overflow Flag Set
BNCX	Branch Extended Negative Flag Clear
BNSX	Branch Extended Negative Flag Set
JMP	Jump
JSR	Jump to Subroutine
NOP	No Operation
RTS	Return from Subroutine
RTI	Return from Interrupt

Branch instruction types include register indirect specifications and program counter (PC) relative address specifications. A branch instruction with a program counter relative address specification can branch within a range around the address that stores the instruction after the branch instruction: -128 to +127 addresses (d8) , -32768 to +32767 addresses (d16), or -8388608 to +8388607 (d24). There are 29 types of relative branch instructions.
Subroutine call instruction types include register indirect specifications and program counter (PC) relative address specifications, and can branch within 16 Mbytes.

If the address calculation of a branch destination falls outside the range x'000000' to x'FFFFFF', then the lower 24 bits of the result will become the branch destination address (wraparound).

4 Memory Space (ROM, RAM)

The memory space of this series can be up to 16 Mbytes. There is no distinction between ROM space and RAM space, so table data can be referenced in programs with data move instructions (MOV instructions). Therefore all addressing modes can be used to access table data, enabling more efficient programming.



5 Addressing Modes

There are six addressing modes. The addressing modes that can be used with each instruction are fixed.

- Register direct addressing
- Immediate addressing
- Register indirect addressing
- Register relative indirect addressing
- Absolute addressing
- Indexed register indirect addressing

The addressing modes provided the six types most frequently used by C compilers. Data move instructions in particular can use all addressing modes. When moving data to or from memory, four addressing modes can be used: register indirect, register relative indirect, absolute, and indexed register indirect.
The indexed register indirect addressing mode allows array data to be accessed efficiently. Calculation instructions can use two addressing modes: register direct and immediate.

This series instruction set is based on 16-bit data access instructions and 24-bit pointer data access instructions. The address specifications for these instructions must be even addresses (word boundaries). Word boundaries must be observed especially for these addressing modes: register indirect, register relative indirect, absolute, and indexed register indirect.
The address specifications for 8-bit data access instructions may also be odd addresses.

■Addressing Modes

Addressing Mode		Address Calculation	Effective Address
Register Direct	Dn An	—	—
Immediate	imm8 imm16 imm24	—	—
Register Indirect	An	23 … 0 [An]	23 … 0 [(24-bit address)]
Register Relative Indirect	(d8, An) :d8 is sign-extended (d16,An) :d16 is sign-extended (d24,An)	23 … 0 [An] + 23 15 7 0 [d24/d16/d8]	23 … 0 [(24-bit address)]
	(d8, PC) :d8 is sign-extended (d16,PC) :d16 is sign-extended (d24,PC) (branch instructions only)	23 … 0 [PC] + 23 15 7 0 [d24/d16/d8]	23 … 0 [(24-bit address)]
Absolute	(abs16) :abs16 is zero-extended (abs24)	23 15 0 [abs24/abs16]	23 … 0 [(24-bit address)]
Indexed Register Indirect	(Dm, An)	23 … 0 [An] + 23 … 0 [Dm]	23 … 0 [(24-bit address)]

5 - 1 Register Direct Addressing

The register direct addressing mode directly specifies a register. The following registers can be specified.

Dn	: data registers (24-bit)
An	: address registers (24-bit)
MDR	: multiply/divide register (16-bit)
PSW	: processor status word (16-bit)

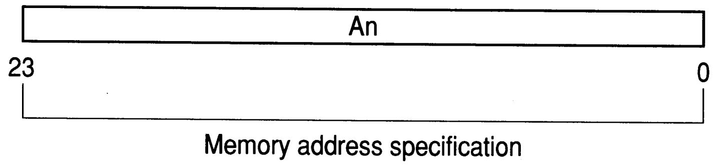
5 - 2 Immediate Addressing

The immediate addressing mode allows move values and mask values to be directly specified as an operand added to the instruction code. Immediate sizes can be 8 bits, 16 bits, and 24 bits.

5 - 3 Register Indirect Addressing

The register indirect addressing mode uses the 24-bit address pointed to by the value in an address register (An).

Register indirect syntax : (A n)

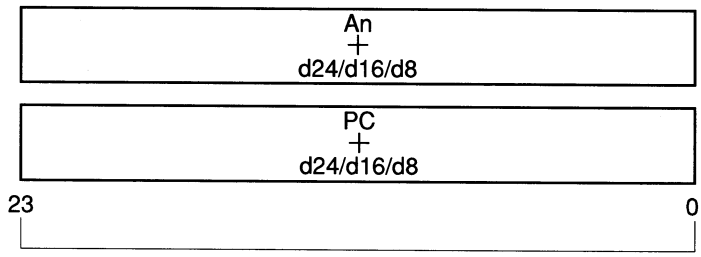


5 - 4 Register Relative Indirect Addressing

The register relative indirect addressing mode is an address specification pointed to by either an address register (An) or the program counter (PC) with a displacement. The size of the displacement can be 8 bits, 16 bits, or 24 bits. An 8-bit or 16-bit displacement will be sign-extended before being added to the address register (An) or program counter (PC).

Register relative indirect syntax:

```
(d8, An)
(d16, An)
(d24, An)
(d8, PC)
(d16, PC)
(d24, PC)
```



Memory address specification

Of the various register relative indirect addressing modes, (d8,PC) is specified in assembly language with a label.

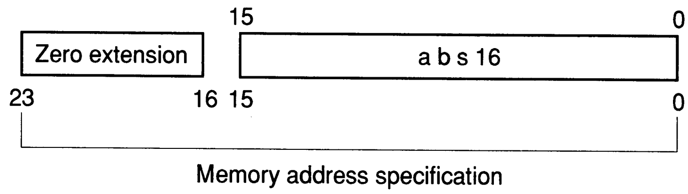
```
        BEQ   LABEL
         :
         :
LABEL   MOV   D0, D1
```

5 - 5 Absolute Addressing

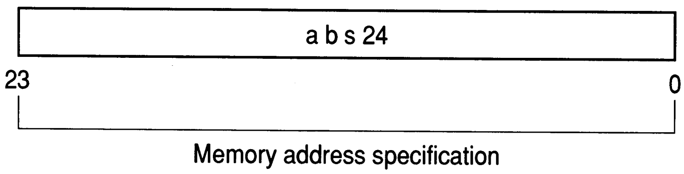
The absolute addressing mode directly specifies a 24-bit address with a 16-bit or 24-bit operand value added to the instruction code. A 16-bit operand will be zero-extended to 24 bits. With this mode data move instructions can directly specify up to 16 Mbytes of memory space (ROM/RAM space).

Absolute syntax: (abs16)
(abs24)

(abs16)



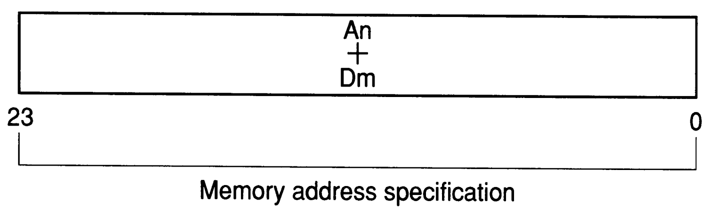
(abs24)



5 - 6 Indexed Register Indirect Addressing

The indexed register indirect addressing mode specifies the address pointed to by an address register (An) and a data register (Dm). The 24-bit value in the address register (An) is added to the 24-bit value in the data register (Dm).

Indexed register indirect syntax: (Dm, An)



*The numbers n and m of An and Dm may be the same.

6 Instruction Format

There are seven instruction formats. The format used by each instruction is fixed.

	8 bits		
Format1	OP		
Format2	OP	imm8/d8	
Format3	OP	imm16 / asb16 / d16	
Format4	OP	OP	
Format5	OP	OP	d8
Format6	OP	OP	imm16 / asb16/ d16
Format7	OP	OP	imm24 / asb24/ d24

OP : instruction operation code

6 - 1 Endian

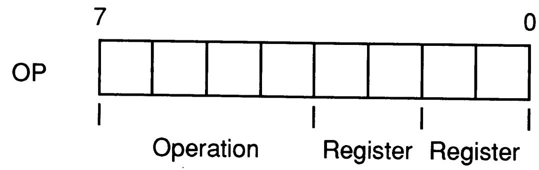
In instruction formats 3, 6, and 7, a 16-bit immediate value (imm16), 16-bit absolute address (abs16), 16-bit displacement (d16), 24-bit immediate value (imm24), 24-bit absolute address (abs24), or 24-bit displacement (d24) may follow after the instruction operation code. In such cases each 8-bit portion appears in order with lower significance placed in a lower address (little-endian).

Ex. The 16-bit immediate value x'1234' is placed in memory as follows for little-endian.

Address 50	x' 34'
Address 51	x' 12'

6-2 Instruction Codes

Instruction formats have either an 8-bit instruction operation code (formats 1~3) or a 16-bit instruction operation code (formats 4~7). In general the lower four bits of the instruction operation code will be coded the register numbers of data registers or address registers. Of those four bits, two bits will represent the register number of the source, and two bits will represent the register number of the destination. The representation of the source and destination register numbers will differ between bits 0/1 and bits 2/3 depending on the instruction.
Instruction format has a 16-bit instruction operation codes enter coded the register number in the second operation.

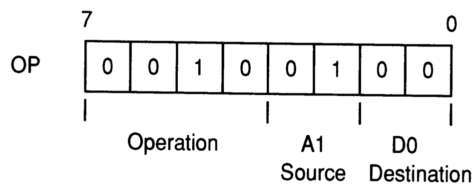


Data registers	D0 :' 00'	
	D1 :' 01'	
	D2 :' 10'	
	D3 :' 11'	
Address registers	A0 :' 00'	
	A1 :' 01'	
	A2 :' 10'	
	A3 :' 11'	

* Other registers are fixed by the instruction.

Ex. The one-byte instruction "MOV (An),Dn" is represented by the instruction code "20+An<<2+Dn".

MOV (A1), D0 ; instruction code x' 24'



---

7 Operand Format

The calculation and move instructions of this series are divided into two different operand formats.

- single operand format (1 operand)
- double operand format (2 operands)

7-1 Single Operand Format (1 Operand)

With single operand format, the register that specifies the operand is read, and then the calculation result is stored back in that register.

Operand Value (**Calculation**) → **Operand Value**

The single operand instructions are **EXT, EXTX, EXTXU, EXTXB, EXTXBU, NOT, ASR, LSR, ROR**, and **ROL**.

The only addressing mode of these instructions is register direct addressing.

7-2 Double Operand Format (2 Operands)

With double operand format, a calculation is performed on the register, RAM, or immediate values specified by the source and destination operands, and then the result is stored back in the register or RAM specified by the destination operand.

[Destination Operand Value] (Calculation) [Source Operand Value] → [Destination Operand Value]

Add the value in register D0 to the value in register D2:

```
ADD     D0, D2
```

(Operation: D2+D0→D2)

The double operand instructions are **MOV**, **MOVX**, **MOVB**, **MOVBU**, **ADD**, **ADDC**, **ADDNF**, **SUB**, **SUBC**, **MUL**, **MULU**, **DIVU**, **CMP**, **AND**, **OR**, **XOR**, **BTST**, **BSET**, and **BCLR**.

The CMP and BTST instructions do not store their calculation results.

MOV instructions have the following six addressing modes.

· Register direct	Dn/An etc.
· Immediate	imm8/imm16/imm24
· Register indirect	(An)
· Register relative indirect	(d8, An)
	(d16, An)
	(d24, An)
· Absolute	(abs16)
	(abs24)
· Indexed register indirect	(Dm, An)

8 Instruction Execution Time

The instruction execution cycle count for this series is given by the shortest number of cycles for an instruction, when instructions already exist in the instruction queue.

【For details, refer to chapter 2, "Instruction Specifications"】
【For details, refer to the appendix, "Instruction Set"】

For faster execution instructions are processed from a 3-level pipeline. Therefore the actual execution cycle count will change depending on the combination of previously executing instructions and memory waits.

Factors causing execution cycles to change:

- Register dependence
  Register dependence occurs when an instruction with register in direct (An) or register relative indirect (d8/d16/d24,An) address ing mode ex-ecutes immediately after that same address register was modified.
- Too few instructions
  (instructions do not exist in the instruction queue)
- Memory waits inserted
- Data bus for external memory is 8 bits wide

Cycle counts will also differ for single-chip mode, memory expansion mode, and processor mode. Specific cycle counts should be measured with an emulator.

Instruction Specifications

2

Symbol Definitions

Following is the list of symbols used in the instruction specifications.

Dn, Dm, Di : Data register (24 bits)
An, Am : Address register (24 bits)
MDR : Multiply/divide register (16 bits)
PSW : Processor status word (16 bits)
PC : Program counter (24 bits)
abs16 : 16-bit absolute address
abs16-l : Low 8 bits of 16-bit absolute address
abs16-h : High 8 bits of 16-bit absolute address
abs24 : 24-bit absolute address
abs24-l : Low 8 bits of 24-bit absolute address
abs24-m : Middle 8 bits of 16-bit absolute address
abs24-h : High 8 bits of 16-bit absolute address
( ) : Memory space specifying address by contents of the parentheses.
imm8 : 8-bit immediate value
imm16 : 16-bit immediate value
imm16-l : Low 8 bits of 16-bit immediate value
imm16-h : High 8 bits of 16-bit immediate value
imm24 : 24-bit immediate value
imm24-l : Low 8 bits of 24-bit immediate value
imm24-m : Middle 8 bits of 24-bit immediate value
imm24-h : High 8 bits of 24-bit immediate value
d8 : 8-bit displacement (-128 to +127 bytes)
d16 : 16-bit displacement (-32768 to +32767 bytes)
d16-l : Low 8 bits of 16-bit displacement
d16-h : High 8 bits of 16-bit displacement
d24 : 24-bit displacement (-8388608 to +8388607 bytes)
d24-l : Low 8 bits of 24-bit displacement
d24-m : Middle 8 bits of 24-bit displacement
d24-h : High 8 bits of 24-bit displacement
.bp : bit position(bit 0 to 23)
.lsb : bit position(bit 0)
.msb : bit position(bit 23)
& : Logical AND
| : Logical OR
^ : Exclusive OR
~ : Bit inversion
<< : Left shift
>> : Right shift
VX : Extended overflow flag
CX : Extended carry flag
NX : Extended negative flag
ZX : Extended zero flag

VF	: Overflow flag
CF	: Carry flag
NF	: Negative flag
ZF	: Zero flag
temp	: CPU internal temporary register
→	: Replacement
···	: Calculation result
label	: Address
mem8(xxx)	: 8-bit data value in memory specified by xxx
mem16(xxx)	: 16-bit data value in memory specified by xxx
mem24(xxx)	: 24-bit data value in memory specified by xxx
●	: Flag may change
—	: Flag will not change
0	: Flag will always be 0
1	: Flag will always be 1
?	: Flag changes undefined
src	: Value of source operand
dest	: Value of destination operand

MOV

Data Move Instruction

MOV Dm, An

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → An

Moves the value of register Dm to register An.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F2 : 30+Dm<<2+An

MOV An, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

An → Dm

Moves the value of register An to register Dm.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F2 : F0+An<<2+Dm

MOV Dn, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dn → Dm

Moves the value of register Dn to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Byte: 1 Cycle: 1 80+Dn<<2+Dm

(!) The same register cannot be specified for Dn and Dm.

MOV An, Am

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

An → Am

Moves the value of register An to register Am.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycles: 2 F2 : 70+An<<2+Am

MOV PSW, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

PSW → Dn

Zero-extends the value of PSW to 24-bits, and moves that value to register Dn.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycles: 2 F3 : F0+Dn

MOV Dn, PSW

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dn → PSW

Moves the lower 16 bits of register Dn to the PSW. The change affects not only the PSW flags VX, CX, NX, ZX, VF, CF, NF, and ZF, but also affects S1, S0, IE, and IM2~IM0.

Flag Changes	Size, Cycles, Codes
VX: Will be set to bit 7 of the moved data. CX: Will be set to bit 6 of the moved data. NX: Will be set to bit 5 of the moved data. ZX: Will be set to bit 4 of the moved data. VF : Will be set to bit 3 of the moved data. CF : Will be set to bit 2 of the moved data. NF : Will be set to bit 1 of the moved data. ZF : Will be set to bit 0 of the moved data.	Bytes: 2 Cycles: 3 F3 : D0+Dn<< 2

MOV MDR, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

MDR → Dn

Zero-extends the value of MDR to 24-bits, and moves that value to register Dn.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycles: 2 F3 : E0+Dn

MOV Dn, MDR

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dn → MDR

Moves the lower 16 bits of register Dn to the MDR.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycles: 2 F3 : C0+Dn<< 2

MOV (An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem16(An) → Dm

Sign-extends the 2-byte value in memory pointed to by register An to 24 bits, and moves that value to register Dm.

Flag Changes

No changes.

Size, Cycles, Codes

Byte: 1
Cycle: 1

20+An<<2+Dm

Accessing to odd addresses in memory is not allowed.

MOV (d8,An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem16(An+d8) → Dm

Adds the value of register An and a sign-extended 8-bit displacement (-128 to +127) to obtain a pointer to memory, then sign-extends the contents of that memory (two bytes) to 24 bits, and moves that value to register Dm.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 2
Cycle: 1

60+An<<2+Dm : d8

Accessing to odd addresses in memory is not allowed.

MOV (d16, An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem16(An+d16) → Dm

Adds the value of register An and a sign-extended 16-bit displacement (-32768 to +32767) to obtain a pointer to memory, then sign-extends the contents of that memory (two bytes) to 24 bits, and moves that value to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 4 Cycles: 2 F7 : C0+An<<2+Dm : d16-l : d16-h

Accessing to odd addresses in memory is not allowed.

MOV (d24,An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem16(An+d24) → Dm

Adds the value of register An and a 24-bit displacement to obtain a pointer to memory, then sign-extends the contents of that memory (two bytes) to 24 bits, and moves that value to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 5 Cycles: 3 F4 : 80+An<<2+Dm : d24-l : d24-m : d24-h

Accessing to odd addresses in memory is not allowed.

MOV (Di,An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem16(An+Di) → Dm

Adds the values of registers An and Di to obtain a pointer to memory, then sign-extends the contents of that memory (two bytes) to 24 bits, and moves that value to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycles: 2 F1 : 40+Di<<4+An<<2+Dm

Accessing to odd addresses in memory is not allowed.

MOV (abs16), Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem16(abs16) → Dn

Zero-extends abs16 to 24 bits to obtain an absolute address pointing to memory, then sign-extends the contents of that memory (two bytes) to 24 bits, and moves that value to register Dn.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycle: 1 C8+Dn : abs16-l : abs16-h

Accessing to odd addresses in memory is not allowed.

MOV (abs24), Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem16(abs24) → Dn

24 bits (abs24) to obtain an absolute address pointing to memory, then sign-extends the contents of that memory (two bytes) to 24 bits, and moves that value to register Dn.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 5
Cycles: 3

F4 : C0+Dn : abs24-l : abs24-m : abs24-h

(!) Accessing to odd addresses in memory is not allowed.

MOV (An), Am

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem24(An) → Am

Moves the value in memory (three bytes) pointed to by register An to register Am.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

70+An<<2+Am: 00

(!) Accessing to odd addresses in memory is not allowed.
This instruction is supported by the assembler.
(The assembler generates the instruction code for "MOV (d8,An),Am"(d8=0).)

MOV (d8, An), Am

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem24(An+d8) → Am

Adds the value of register An and a sign-extended 8-bit displacement (-128 to +127) to obtain a pointer to memory, and moves that value in memory (three bytes) to register Am.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycles: 2 70+An<<2+Am : d8

Accessing to odd addresses in memory is not allowed.

MOV (d16, An), Am

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem24(An+d16) → Am

Adds the value of register An and a sign-extended 16-bit displacement (-32768 to +32767) to obtain a pointer to memory, and moves that value in memory (three bytes) to register Am.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 4 Cycles: 3 F7 : B0+An<<2+Am : d16-l : d16-h

Accessing to odd addresses in memory is not allowed.

MOV (d24,An), Am	VX	CX	NX	ZX	VF	CF	NF	ZF
	–	–	–	–	–	–	–	–

Operation

mem24(An+d24) → Am

Adds the value of register An and the 24-bit displacement to obtain a pointer to memory, and moves that value in memory (three bytes) to register Am.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 5 Cycles: 4 F4 : F0+An<<2+Am : d24-l : d24-m : d24-h

Accessing to odd addresses in memory is not allowed.

MOV (abs16), An

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem24(abs16) → An

Zero-extends abs16 to 24 bits to obtain an absolute address pointing to memory, and moves that value in memory (three bytes) to register An.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 4 Cycles: 3 F7 : 30+An : abs16-l : abs16-h

Accessing to odd addresses in memory is not allowed.

MOV (abs24), An

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem24(abs24) → An

24 bits to obtain an absolute address pointing to memory, and moves that value in memory (three bytes) to register An.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 5 Cycles: 4 F4 : D0+An : abs24-l : abs24-m : abs24-h

Accessing to odd addresses in memory is not allowed.

MOV Dm, (An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem16(An)

Moves the lower 16 bits of register Dm (two bytes) to the memory pointed to by register An.

Flag Changes	Size, Cycles, Codes
No changes.	Byte: 1 Cycle: 1 00+An<<2+Dm

Accessing to odd addresses in memory is not allowed.

MOV Dm, (d8,An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem16(An+d8)

Adds the value of register An and a sign-extended 8-bit displacement (-128 to +127) to obtain a pointer to memory, and moves the lower 16 bits of register Dm (two bytes) to that memory.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycle: 1 40+An<<2+Dm : d8

Accessing to odd addresses in memory is not allowed.

MOV Dm, (d16,An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem16(An+d16)

Adds the value of register An and a sign-extended 16-bit displacement (-32768 to +32767) to obtain a pointer to memory, and moves the lower 16 bits of register Dm (two bytes) to that memory.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 4 Cycles: 2 F7 : 80+An<<2+Dm : d16-l : d16-h

(!) Accessing to odd addresses in memory is not allowed.

MOV Dm, (d24,An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem16(An+d24)

Adds the value of register An and a 24-bit displacement to obtain a pointer to memory, and moves the lower 16 bits of register Dm (two bytes) to that memory.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 5 Cycles: 3 F4 : 00+An<<2+Dm : d24-l :d24-m : d24-h

(!) Accessing to odd addresses in memory is not allowed.

MOV Dm, (Di, An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem16(An+Di)

Adds the value of register An and Di to obtain a pointer to memory, and moves the lower 16 bits of register Dm (two bytes) to that memory.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycles: 2 F1 : C0+Di<<4+An<<2+Dm

(!) Accessing to odd addresses in memory is not allowed.

MOV Dn, (abs16)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dn → mem16(abs16)

Zero-extends abs16 to 24 bits to obtain an absolute address pointing to memory, and moves the lower 16 bits of register Dn (two bytes) to that memory.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycle: 1 C0+Dn : abs16-l : abs16-h

(!) Accessing to odd addresses in memory is not allowed.

MOV Dn, (abs24)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dn → mem16(abs24)

Moves the lower 16 bits of register Dn (two bytes) to the memory pointed to by the 24-bit absolute address abs24.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 5 Cycles: 3 F4 : 40+Dn : abs24-l : abs24-m : abs24-h

Accessing to odd addresses in memory is not allowed.

MOV Am, (An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Am → mem24(An)

Moves the value of register Am (three bytes) to the memory pointed to by the register An.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycles: 2 50+An<<2+Am : 00

Accessing to odd addresses in memory is not allowed.
This instruction is supported by the assembler.
(The assembler generates the instruction code for "MOV Am, (d8,An)"(d8=0).)

MOV Am, (d8, An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Am → mem24(An+d8)

Adds the value of register An and a sign-extended 8-bit displacement (-128 to +127) to obtain a pointer to memory, and moves the 3-byte value of register Am to that memory.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycles: 2 50+An<<2+Am : d8

Accessing to odd addresses in memory is not allowed.

MOV Am, (d16, An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Am → mem24(An+d16)

Adds the value of register An and a sign-extended 16-bit displacement (-32768 to +32767) to obtain a pointer to memory, and moves the 3-byte value of register Am to that memory.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 4 Cycles: 3 F7 : A0+An<<2+Am : d16-l : d16-h

Accessing to odd addresses in memory is not allowed.

MOV Am, (d24, An)

VX	CX	NX	ZX	VF	CF	NF	ZF
–	–	–	–	–	–	–	–

Operation

Am → mem24(An+d24)

Adds the value of register An and a 24-bit displacement to obtain a pointer to memory, and moves the 3-byte of register Am to that memory.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 5 Cycles: 4 F4 : 10+An<<2+Am : d24-l : d24-m : d24-h

Accessing to odd addresses in memory is not allowed.

MOV An, (abs16)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

An → mem24(abs16)

Zero-extends abs16 to 24 bits to obtain an absolute address pointing to memory, and moves the 3-byte value of register An to that memory.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 4

Cycles: 3

F7 : 20+An : abs16-l : abs16-h

Accessing to odd addresses in memory is not allowed.

MOV An, (abs24)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

An → mem24(abs24)

Moves the 3-byte value of register An to memory pointed to by the 24-bit absolute address abs24.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 5

Cycles: 4

F4 : 50+An : abs24-l : abs24-m : abs24-h

Accessing to odd addresses in memory is not allowed.

MOV imm8, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

imm8 → Dn

Sign-extends the 8-bit immediate value imm8 to 24 bits, and moves it to register Dn.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycle: 1 80+Dn<<2+Dn : imm8

(!) The 8-bit immediate value imm8 will be sign-extended to 24 bits.

MOV imm16, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

imm16 → Dn

Sign-extends the 16-bit immediate value imm16 to 24 bits, and moves it to register Dn.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycle: 1 F8+Dn : imm16-l : imm16-h

(!) The 16-bit immediate value imm16 will be sign-extended to 24 bits.

MOV imm24, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

imm24 → Dn

Moves the 24-bit immediate value imm24 to register Dn.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 5
Cycles: 3

F4 : 70+Dn : imm24-l : imm24-m : imm24-h

MOV imm16, An

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

imm16 → An

Zero-extends the 16-bit immediate value imm16 to 24 bits, and moves it to register An.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 3
Cycle: 1

DC+An : imm16-l : imm16-h

(!) The 16-bit immediate value imm16 will be zero-extended to 24 bits.

MOV imm24, An

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

imm24 → An

Moves the 24-bit immediate value imm24 to register An.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 5

Cycles: 3

F4 : 74+An : imm24-l : imm24-m : imm24-h

MOVX

Data move instruction

MOVX (d8, An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem24 (An+d8)→ Dm

Sign-extends the 8-bit displacement (-128 to +127), adds it to register An to obtain a pointer to memory, and moves the contents of that memory (3 bytes) to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 F5 : 70+An<<2+Dm : d8

(!) Accessing to odd addresses in memory is not allowed.

MOVX (d16, An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem24(An+d16) → Dm

Sign-extends the 16-bit displacement (-32768 to +32767), adds it to register An to obtain a pointer to memory, and moves the contents of that memory (3 bytes) to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 4 Cycles: 3 F7 : 70+An<<2+Dm : d16-l : d16-h

(!) Accessing to odd addresses in memory is not allowed.

MOVX (d24, An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem24(An+d24) → Dm

Adds the 24-bit displacement to register An to obtain a pointer to memory, and moves the contents of that memory (3 bytes) to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 5 Cycles: 4 F4 : B0+An<<2+Dm : d24-l : d24-m : d24-h

Accessing to odd addresses in memory is not allowed.

MOVX Dm, (d8, An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem24 (An+d8)

Adds the value of register An and a sign-extended 8-bit displacement (-128 to +127) to obtain a pointer to memory, and moves the 3-byte value of register Dm to that memory.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 F5 : 50+An<<2+Dm : d8

Accessing to odd addresses in memory is not allowed.

MOVX Dm, (d16, An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem24(An+d16)

Adds the value of register An and a sign-extended 16-bit displacement (-32768 to +32767) to obtain a pointer to memory, and moves the 3-byte value of register Dm to that memory.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 4
Cycles: 3

F7 : 60+An<<2+Dm : d16-l : d16-h

Accessing to odd addresses in memory is not allowed.

MOVX Dm, (d24, An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem24(An+d24)

Adds the value of register An and a 24-bit displacement to obtain a pointer to memory, and moves the 3-byte value of register Dm to that memory.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 5
Cycles: 4

F4 : 30+An<<2+Dm : d24-l : d24-m : d24-h

Accessing to odd addresses in memory is not allowed.

MOVB Data move instruction

MOVB (An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(An) → Dm

Sign-extends the one-byte contents of memory pointed to by register An to 24 bits, and moves the result to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycles: 2 30+An<<2+Dm : B8+Dm

This instruction is supported by the assembler (the assembler generates the two instruction codes for "MOVBU (An),Dm" and "EXTXB Dm").

MOVB (d8, An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(An+d8) → Dm

Sign-extends the 8-bit displacement (-128 to +127), adds it to register An to obtain a pointer to memory, sign-extends the contents of that memory (1 byte) to 24 bits, and moves the result to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 2 F5 : 20+An<<2+Dm : d8

MOVB (d16, An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(An+d16) → Dm

Sign-extends the 16-bit displacement (-32768 to +32767), adds it to register An to obtain a pointer to memory, sign-extends the contents of that memory (1 byte) to 24 bits, and moves the result to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 4 Cycles: 2 F7 : D0+An<<2+Dm : d16-l : d16-h

MOVB (d24, An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(An+d24) → Dm

Adds the 24-bit displacement to register An to obtain a pointer to memory, sign-extends the contents of that memory (1 byte) to 24 bits, and moves the result to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 5 Cycles: 3 F4 : A0+An<<2+Dm : d24-l : d24-m : d24-h

MOVB (Di, An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(An+Di) → Dm

Adds the register Di to register An to obtain a pointer to memory, sign-extends the contents of that memory (1 byte) to 24 bits, and moves the result to register Dm.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 2

Cycles: 2

F0 : 40+Di<<4+An<<2+Dm

MOVB (abs16), Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(abs16) → Dn

Zero-extends abs16 to 24 bits to obtain an absolute address pointer to memory, sign-extends the contents of that memory (1 byte) to 24 bits, and moves the result to register Dn.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 4

Cycles: 2

CC+Dn : abs16-l : abs16-h : B8+Dn

This instruction is supported by the assembler (the assembler generates the two instruction codes for "MOVBU (abs16),Dn" and "EXTXB Dn").

MOVB (abs24), Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(abs24) → Dn

Sign-extends the one-byte contents of memory pointed to by 24 bits (abs24) to obtain an absolute address pointer to memory, and moves the result to register Dn.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 5 Cycles: 3 F4 : C4+Dn : abs24-l : abs24-m : abs24-h

MOVB Dm, (An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem8(An)

Moves the lower 8 bits of register Dm (one byte) to the memory pointed to by register An.

Flag Changes	Size, Cycles, Codes
No changes.	Byte: 1 Cycle: 1 10+Dm<<2+An

MOVB Dm, (d8, An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem8(An+d8)

Sign-extends the 8-bit displacement (-128 to +127), adds it to register An to obtain a pointer to memory, and moves the lower 8 bits of register Dm (one byte) to that memory.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 2 F5 : 10+An<<2+Dm : d8

MOVB Dm, (d16, An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem8(An+d16)

Sign-extends the 16-bit displacement (-32768 to +32767), adds it to register An to obtain a pointer to memory, and moves the lower 8 bits of register Dm (one byte) to that memory.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 4 Cycles: 2 F7 : 90+An<<2+Dm : d16-l : d16-h

MOVB Dm, (d24, An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem8(An+d24)

Adds the 24-bit displacement and register An to obtain a pointer to memory, and moves the lower 8 bits of register Dm (one byte) to that memory.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 5
Cycles: 3

F4 : 20+An<<2+Dm : d24-l : d24-m : d24-h

MOVB Dm, (Di, An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dm → mem8(An+Di)

Adds register An and register Di to obtain a pointer to memory, and moves the lower 8 bits of register Dm (one byte) to that memory.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F0 : C0+Di<<4+An<<2+Dm

MOVB Dn, (abs16)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dn → mem8(abs16)

Zero-extends abs16 to 24 bits to obtain an absolute address pointing to memory, and moves the lower 8 bits of register Dn (one byte) to that memory.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 3
Cycle: 1

C4+Dn : abs16-l : abs16-h

MOVB Dn, (abs24)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

Dn → mem8(abs24)

Moves the lower 8 bits of register Dn (one byte) to the memory pointed to by the 24-bit absolute address abs24.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 5
Cycles: 3

F4 : 44+Dn : abs24-l : abs24-m : abs24-h

MOVBU

Data move instruction

MOVBU (An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(An) → Dm

Zero-extends the 1-byte contents of memory pointed to by register An to 24 bits, and stores the result in register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Byte: 1 Cycle: 1 30+An<<2+Dm

MOVBU (d8, An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(An+d8) → Dm

Sign-extends the 8-bit displacement (-128 to +127), adds it to register An to obtain a pointer to memory, zero-extends the contents of that memory (1-byte) to 24 bits, and moves the result to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 2 F5 : 30+An<<2+Dm : d8

MOVBU (d16, An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(An+d16) → Dm

Sign-extends the 16-bit displacement (-32768 to +32767), adds it to register An to obtain a pointer to memory, zero-extends the contents of that memory (1-byte) to 24 bits, and moves the result to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 4 Cycles: 2 F7 : 50+An<<2+Dm : d16-l : d16-h

MOVBU (d24, An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(An+d24) → Dm

Adds the 24-bit displacement to register An to obtain a pointer to memory, zero-extends the contents of that memory (1-byte) to 24 bits, and moves the result to register Dm.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 5 Cycles: 3 F4 : 90+An<<2+Dm : d24-l : d24-m : d24-h

MOVBU (Di, An), Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(An+Di) → Dm

Adds register Di to register An to obtain a pointer to memory, zero-extends the contents of that memory (1-byte) to 24 bits, and moves the result to register Dm.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 2

Cycles: 2

F0 : 80+Di<<4+An<<2+Dm

MOVBU (abs16), Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(abs16) → Dn

Zero-extends abs16 to 24 bits to obtain an absolute address pointing to memory, zero-extends the contents of that memory (1-byte) to 24-bits, and moves the result to register Dn.

Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 3

Cycle: 1

CC+Dn : abs16-l : abs16-h

MOVBU (abs24), Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem8(abs24) → Dn

Zero-extends the contents of 24 bits to obtain an absolute address pointing to memory (1-byte) to 24-bits, and moves the result to register Dn.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 5 Cycles: 3 F4 : C8+Dn : abs24-l : abs-24-m : abs24-h

EXT Data move instruction

EXT Dn	VX	CX	NX	ZX	VF	CF	NF	ZF
	—	—	—	—	—	—	—	—

Operation

IF Dn.bp15 = 0, then x'0000'→ MDR
IF Dn.bp15 = 1, then x'FFFF'→ MDR

Sign-extends the lower 16 bits of register Dn (2 bytes) to 32 bits, and moves the 16-bit extension to register MDR. The contents of Dn are not changed.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycles: 3 F3 : C1+Dn<<2

EXTX

Data move instruction

EXTX Dn	VX	CX	NX	ZX	VF	CF	NF	ZF
	—	—	—	—	—	—	—	—

Operation

IF Dn.bp15 = 0, then Dn& x'00FFFF' → Dn
IF Dn.bp15 = 1, then Dn I x'FF0000' → Dn

Sign-extends the lower 16 bits of register Dn (2 bytes) to 24 bits, and stores the result in register Dn.

Flag Changes	Size, Cycles, Codes
No changes.	Byte: 1 Cycle: 1 B0+Dn

EXTXU

Data move instruction

EXTXU Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation	
	Dn&x'00FFFF' → Dn Zero-extends the lower 16 bits of register Dn (2 bytes) to 24 bits, and stores the result in register Dn.

Flag Changes	Size, Cycles, Codes
No changes.	Byte: 1 Cycle: 1 B4+Dn

EXTXB

Data move instruction

EXTXB Dn	VX	CX	NX	ZX	VF	CF	NF	ZF
	—	—	—	—	—	—	—	—

Operation

IF Dn.bp7 = 0, then Dn&x'0000FF' → Dn
IF Dn.bp7 = 1, then Dn I x'FFFF00' → Dn

Sign-extends the lower 8 bits of register Dn (1 byte) to 24 bits, and stores the result in register Dn.

Flag Changes	Size, Cycles, Codes
No changes.	Byte: 1 Cycle: 1 B8+Dn

EXTXBU

Data move instruction

EXTXBU Dn	VX	CX	NX	ZX	VF	CF	NF	ZF
	—	—	—	—	—	—	—	—

Operation	
	Dn&x'0000FF' → Dn Zero-extends the lower 8 bits of register Dn (1 byte) to 24 bits, and stores the result in register Dn.

Flag Changes	Size, Cycles, Codes
No changes.	Byte: 1 Cycle: 1 BC+Dn

ADD

Arithmetic Calculation Instructions

ADD Dn, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dm+Dn → Dm

Adds the value of register Dm to register Dn, and stores the result in register Dm.

Flag Changes	Size, Cycles, Codes
VX: Set if result taken as a 24-bit signed value overflows; reset otherwise. CX: Set if a carry is generated from the MSB; reset otherwise. NX: Set if the MSB of the result is 1; reset otherwise. ZX: Set if the result is 0; reset otherwise. VF: Set if result taken as a 16-bit signed value overflows; reset otherwise. CF: Set if a carry is generated from bit 15; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Byte: 1 Cycle: 1 90+Dn<<2+Dm

ADD Dm, An

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

An+Dm → An

Adds the value of register An to register Dm, and stores the result in register An.

Flag Changes	Size, Cycles, Codes
VX: Set if result taken as a 24-bit signed value overflows; reset otherwise. CX: Set if a carry is generated from the MSB; reset otherwise. NX: Set if the MSB of the result is 1; reset otherwise. ZX: Set if the result is 0; reset otherwise. VF: Set if result taken as a 16-bit signed value overflows; reset otherwise. CF: Set if a carry is generated from bit 15; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 2 Cycles: 2 F2 : 00+Dm<<2+An

ADD An, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dm+An → Dm

Adds the value of register Dm to register An, and stores the result in register Dm.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a carry is generated from the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a carry is generated from bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F2 : C0+An<<2+Dm

ADD An, Am

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Am+An → Am

Adds the value of register Am to register An, and stores the result in register Am.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a carry is generated from the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a carry is generated from bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F2 : 40+An<<2+Am

ADD imm8, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dn+imm8 → Dn

Sign-extends the 8-bit immediate value imm8 (-128 to +127) to 24 bits, adds it to register Dn, and stores the result in register Dn.

Flag Changes	Size, Cycles, Codes
VX: Set if result taken as a 24-bit signed value overflows; reset otherwise. CX: Set if a carry is generated from the MSB; reset otherwise. NX: Set if the MSB of the result is 1; reset otherwise. ZX: Set if the result is 0; reset otherwise. VF: Set if result taken as a 16-bit signed value overflows; reset otherwise. CF: Set if a carry is generated from bit 15; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 2 Cycle: 1 D4+Dn : imm8

The 8-bit immediate value imm8 will be sign-extended to 24 bits.

ADD imm16, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dn+imm16 → Dn

Sign-extends the 16-bit immediate value imm16 (-32768 to +32767) to 24 bits, adds it to register Dn, and stores the result in register Dn.

Flag Changes	Size, Cycles, Codes
VX: Set if result taken as a 24-bit signed value overflows; reset otherwise. CX: Set if a carry is generated from the MSB; reset otherwise. NX: Set if the MSB of the result is 1; reset otherwise. ZX: Set if the result is 0; reset otherwise. VF: Set if result taken as a 16-bit signed value overflows; reset otherwise. CF: Set if a carry is generated from bit 15; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 4 Cycles: 2 F7 :18+Dn : imm16-l : imm16-h

The16-bit immediate value imm16 will be sign-extended to 24 bits.

ADD imm24, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dn+imm24 → Dn

Adds the 24-bit immediate value imm24 to register Dn, and stores the result in register Dn.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a carry is generated from the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a carry is generated from bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 5
Cycles: 3

F4 : 60+Dn : imm24-l : imm24-m : imm-h

ADD imm8, An

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

An+imm8 → An

Sign-extends the 8-bit immediate value imm8 (-128 to +127) to 24 bits, adds it to register An, and stores the result in register An.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a carry is generated from the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a carry is generated from bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycle: 1

D0+An : imm8

(!) The 8-bit immediate value imm8 will be sign-extended to 24 bits.

ADD imm16, An

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

An+imm16 → An

Sign-extends the 16-bit immediate value imm16 (-32768 to +32767) to 24 bits, adds it to register An, and stores the result in register An.

Flag Changes	Size, Cycles, Codes
VX: Set if result taken as a 24-bit signed value overflows; reset otherwise. CX: Set if a carry is generated from the MSB; reset otherwise. NX: Set if the MSB of the result is 1; reset otherwise. ZX: Set if the result is 0; reset otherwise. VF: Set if result taken as a 16-bit signed value overflows; reset otherwise. CF: Set if a carry is generated from bit 15; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 4 Cycles: 2 F7 : 08+An : imm16-l : imm16-h

(!) The 16-bit immediate value imm16 will be sign-extended to 24 bits.

ADD imm24, An

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

An+imm24 → An

Adds the value of register An to the 24-bit immediate value, and stores the result in register An.

Flag Changes	Size, Cycles, Codes
VX: Set if result taken as a 24-bit signed value overflows; reset otherwise. CX: Set if a carry is generated from the MSB; reset otherwise. NX: Set if the MSB of the result is 1; reset otherwise. ZX: Set if the result is 0; reset otherwise. VF: Set if result taken as a 16-bit signed value overflows; reset otherwise. CF: Set if a carry is generated from bit 15; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 5 Cycles: 3 F4 : 64+An : imm24-l : imm24-m : imm24-h

ADDC

Arithmetic Calculation Instructions

ADDC Dn, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dm+Dn+CF → Dm

Adds the value of register Dn and the 16-bit data carry (CF) to register Dm, and stores the result in register Dm.

Flag Changes	Size, Cycles, Codes
VX: Set if result taken as a 24-bit signed value overflows; reset otherwise. CX: Set if a carry is generated from the MSB; reset otherwise. NX: Set if the MSB of the result is 1; reset otherwise. ZX: Set if the result is 0; reset otherwise. VF: Set if result taken as a 16-bit signed value overflows; reset otherwise. CF: Set if a carry is generated from bit 15; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if ZF=1 and the lower 16 bits of the result are 0; reset otherwise.	Bytes: 2 Cycles: 2 F2 : 80+Dn<<2+Dm

(!) 32-bit addition can be performed by combining an ADD instruction and ADDC instruction. A zero-check on the 32-bit result can be done only with ZF after the ADDC instruction has been executed.

ADDNF Arithmetic Calculation Instructions

ADDNF imm8, An

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

An+imm8 → An

Sign-extends the 8-bit immediate value imm8 (-128 to +127) to 24 bits, adds it to register An, and stores the result in register An.

Flag Changes	Size, Cycles, Codes
No changes	Bytes: 3 Cycles: 2 F5 : 0C+An : imm8

The 8-bit immediate value is sign-extended to 24 bits.
Flags do not change.
The primary application for this instruction is incrementing the stack pointer (A3) past the stack frame at the end of a subroutine without changing the PSW flags reporting results. See Section 1.2 in Chapter 3 "Use of Instructions" for an example.

SUB

Arithmetic Calculation Instructions

SUB Dn, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dm-Dn → Dm

Subtracts the value of register Dn from the value of register Dm, and stores the result in register Dm.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated from the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated from bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Byte: 1
Cycle: 1

A0+Dn<<2+Dm

SUB Dm, An

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

An-Dm → An

Subtracts the value of register Dm from the value of register An, and stores the result in register An.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated from the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated from bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F2 : 10+Dn<<2+An

SUB An, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dm-An → Dm

Subtracts the value of register An from the value of register Dm, and stores the result in register Dm.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated from the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated from bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F2 : D0+An<<2+Dm

SUB An, Am

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Am-An → Am

Subtracts the value of register An from the value of register Am, and stores the result in register Am.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated from the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated from bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F2 : 50+An<<2+Am

SUB imm16, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dn-imm16 → Dn

Sign-extends the 16-bit immediate value imm16 (-32768 to +32767) to 24 bits, subtracts it from register Dn, and stores the result in register Dn.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated from the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated from bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 4
Cycles: 2

F7 : 1C+Dn : imm16-l : imm16-h

(!) The 16-bit immediate value imm16 will be sign-extended to 24 bits.

SUB imm24, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dn-imm24 → Dn

Subtracts the 24-bit immediate value from the value of register Dn, and stores the result in register Dn.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated from the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated from bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 5
Cycles: 3
F4 : 68+Dn : imm24-l : imm24-m : imm24-h

SUB imm16, An

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

An-imm16 → An

Sign-extends the 16-bit immediate value imm16 to 24 bits (-32768 to +32767), subtracts it from register An, and stores the result in register An.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated from the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated from bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 4
Cycles: 2

F7 : 0C+An : imm16-l : imm16-h

(!) The 16-bit immediate value imm16 will be sign-extended to 24 bits.

SUB imm24, An

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

An-imm24 → An

Subtracts the 24-bit immediate value from the value of register An, and stores the result in register An.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated from the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated from bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 5
Cycles: 3

F4 : 6C+An : imm24-l : imm24-m : imm24-h

SUBC

Arithmetic Calculation Instructions

SUBC Dn, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dm-Dn-CF → Dm

Subtracts the value of register Dn and the 16-bit data borrow (CF) from register Dm, and stores the result in register Dm.

Flag Changes	Size, Cycles, Codes
VX: Set if result taken as a 24-bit signed value overflows; reset otherwise. CX: Set if a borrow is generated from the MSB; reset otherwise. NX: Set if the MSB of the result is 1; reset otherwise. ZX: Set if the result is 0; reset otherwise. VF: Set if result taken as a 16-bit signed value overflows; reset otherwise. CF: Set if a borrow is generated from bit 15; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if ZF=1 and the lower 16 bits of the result are 0; reset otherwise.	Bytes: 2 Cycles: 2 F2 : 90+Dn<<2+Dm

(!) 32-bit subtraction can be performed by combining an SUB instruction and SUBC instruction. A zero-check on the 32-bit result can be done only with ZF after the SUBC instruction has been executed.

MUL Arithmetic Calculation Instructions

MUL Dn, Dm

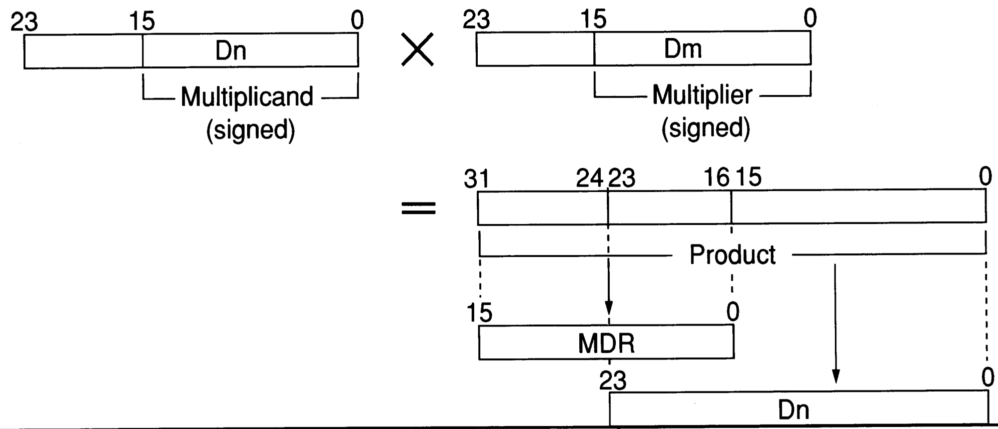
VX	CX	NX	ZX	VF	CF	NF	ZF
?	?	?	?	0	?	●	●

Operation

Dm * Dn → Dm
(Dm * Dn)>>16 → MDR

Multiplies the lower 16 bits of register Dm and the lower 16 bits of register Dn as signed numbers, and stores the upper 16 bits of the 32-bit result in register MDR and the lower 24 bits in register Dm. The upper 8 bits of register Dm will be the same as the lower 8 bits of register MDR.

The MUL instruction performs a signed multiplication 16 bits x 16 bits = 32 bits.



Flag Changes

VX, CX, NX, ZX: Undefined.
VF: Always 0.
CF: Undefined.
NF: Set if MSB of result (32 bits) is 1; reset otherwise.
ZF: Set if result is 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 12

F3 : 40+Dn<<2+Dm

MULU

Arithmetic Calculation Instructions

MULU Dn, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
?	?	?	?	0	?	●	●

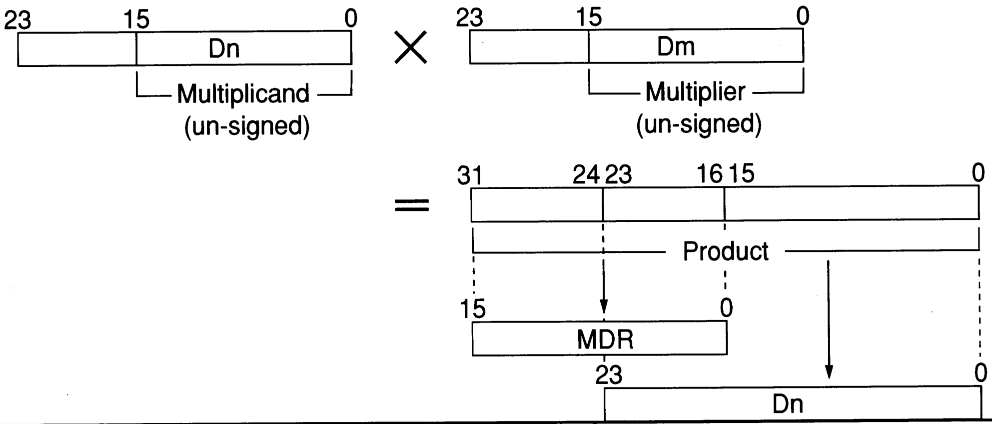
Operation

Dm * Dn → Dm

(Dm * Dn)>>16 → MDR

Multiplies the lower 16 bits of register Dm and the lower 16 bits of register Dn as unsigned numbers, and stores the upper 16 bits of the 32-bit result in register MDR and the lower 24 bits in register Dm. The upper 8 bits of register Dm will be the same as the lower 8 bits of register MDR.

The MULU instruction performs an unsigned multiplication 16 bits x 16 bits = 32 bits.



Flag Changes

VX, CX, NX, ZX: Undefined.
VF: Always 0.
CF: Undefined.
NF: Set if MSB of result (32 bits) is 1; reset otherwise.
ZF: Set if result is 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2

Cycles: 12

F3 : 50+Dn<<2+Dm

DIVU

Arithmetic Calculation Instructions

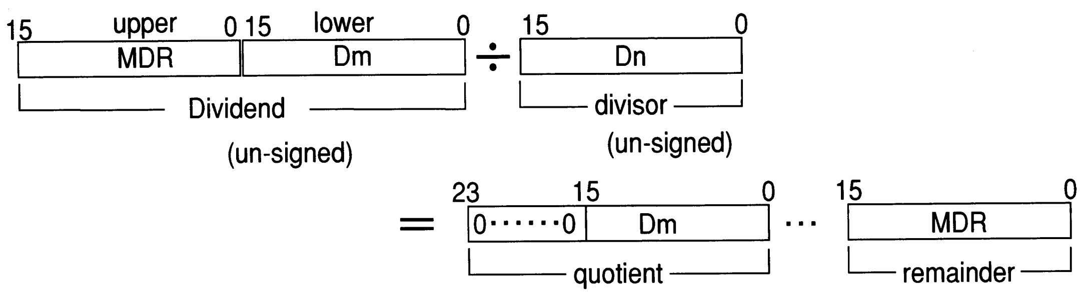
DIVU Dn, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
?	?	0/?	●/?	0/1	?	●/?	●/?

Operation

(MDR<<16+Dm)/Dn→ Dm ··· MDR

Divides the unsigned 32-bit concatenation of register MDR's upper 16 bits and register Dm's lower 16 bits by the unsigned 16-bit value from the lower 16 bits of register Dn. The 16-bit quotient is zero-extended to 24 bits and stored in register Dm. The 16-bit remainder is stored in register MDR. If an overflow occurs (VF=1), then the resulting contents of registers Dm and MDR will be undefined. VF will also be set to 1 when division by zero is performed (divisor is 0).
The DIVU instruction performs an unsigned division 32 bits / 16 bits = 16 bits...16bits.



Flag Changes	Size, Cycles, Codes
● If VF = 0 (Indicates that no overflow occurs for 16-bit unsigned quotient) VX,CX: Undefined. NX: 0 (indicates that the MSB of the quotient in register Dm is 0). ZX: Set if quotient in register Dm is 0; reset otherwise. VF: 0 CF: Undefined. NF: Set if MSB of 16-bit quotient is 1; reset otherwise. ZF: Set if 16-bit quotient is 0; reset otherwise. ● If VF = 1 (Indicates that the quotient overflowed or that division by zero was performed) VX, CX, NX, ZX: Undefined. VF: 1 CF, NF, ZF: Undefined.	Bytes: 2 Cycles: 13 F3 : 60+Dn<<2+Dm

CMP

Arithmetic Calculation Instructions

CMP Dn, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dm-Dn

Subtracts the values of register Dn from the value of register Dm, but the result is not stored in register Dm.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated to the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated to bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F3 : 90+Dn<<2+Dm

CMP Dm, An

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

An-Dm

Subtracts the values of register Dm from the value of register An, but the result is not stored in register An.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated to the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated to bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F2 : 20+Dm<<2+An

CMP An, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dm-An

Subtracts the values of register An from the value of register Dm, but the result is not stored in register Dm.

Flag Changes	Size, Cycles, Codes
VX: Set if result taken as a 24-bit signed value overflows; reset otherwise. CX: Set if a borrow is generated to the MSB; reset otherwise. NX: Set if the MSB of the result is 1; reset otherwise. ZX: Set if the result is 0; reset otherwise. VF: Set if result taken as a 16-bit signed value overflows; reset otherwise. CF: Set if a borrow is generated to bit 15; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 2 Cycles: 2 F2 : E0+An<<2+Dm

CMP An, Am

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Am-An

Subtracts the values of register An from the value of register Am, but the result is not stored in register Am.

Flag Changes	Size, Cycles, Codes
VX: Set if result taken as a 24-bit signed value overflows; reset otherwise. CX: Set if a borrow is generated to the MSB; reset otherwise. NX: Set if the MSB of the result is 1; reset otherwise. ZX: Set if the result is 0; reset otherwise. VF: Set if result taken as a 16-bit signed value overflows; reset otherwise. CF: Set if a borrow is generated to bit 15; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 2 Cycles: 2 F2 : 60+An<<2+Am

CMP imm8, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dn-imm8

Sign-extends the 8-bit immediate value imm8 to 24-bits, subtracts it from register Dn, but does not store the result in register Dn.

Flag Changes	Size, Cycles, Codes
VX: Set if result taken as a 24-bit signed value overflows; reset otherwise. CX: Set if a borrow is generated to the MSB; reset otherwise. NX: Set if the MSB of the result is 1; reset otherwise. ZX: Set if the result is 0; reset otherwise. VF: Set if result taken as a 16-bit signed value overflows; reset otherwise. CF: Set if a borrow is generated to bit 15; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 2 Cycle: 1 D8+Dn : imm8

(!) The 8-bit immediate value imm8 will be sign-extended to 24 bits.

CMP imm16, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dn-imm16

Sign-extends the 16-bit immediate value imm16 to 24-bits, subtracts it from register Dn, but does not store the result in register Dn.

Flag Changes	Size, Cycles, Codes
VX: Set if result taken as a 24-bit signed value overflows; reset otherwise. CX: Set if a borrow is generated to the MSB; reset otherwise. NX: Set if the MSB of the result is 1; reset otherwise. ZX: Set if the result is 0; reset otherwise. VF: Set if result taken as a 16-bit signed value overflows; reset otherwise. CF: Set if a borrow is generated to bit 15; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 4 Cycles: 2 F7 : 48+Dn : imm16-l : imm16-h

(!) The 16-bit immediate value imm16 will be sign-extended to 24 bits.

CMP imm24, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

Dn-imm24

Subtracts the 24-bit immediate value imm24 from register Dn, but does not store the result in register Dn.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated to the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated to bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 5
Cycles: 3

F4 : 78+Dn : imm24-l : imm24-m : imm24-h

CMP imm16, An

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

An-imm16

Zero-extends the 16-bit immediate value imm16 to 24 bits, subtracts it from register An,but does not store the result in register An.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated to the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated to bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 3
Cycle: 1

EC+An : imm16-l : imm16-h

(!) The 16-bit immediate value imm16 will be zero-extended to 24 bits.

CMP imm24, An

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

An-imm24

Subtracts the 24-bit immediate value imm24 from register An, but does not store the result in register An.

Flag Changes

VX: Set if result taken as a 24-bit signed value overflows; reset otherwise.
CX: Set if a borrow is generated to the MSB; reset otherwise.
NX: Set if the MSB of the result is 1; reset otherwise.
ZX: Set if the result is 0; reset otherwise.
VF: Set if result taken as a 16-bit signed value overflows; reset otherwise.
CF: Set if a borrow is generated to bit 15; reset otherwise.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 5
Cycles: 3

F4 : 7C+An : imm24-l : imm24-m : imm24-h

AND

Logical Calculation Instructions

AND Dn, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	●	●

Operation

Dm&(x'FF0000' | Dn) → Dm

Performs a bitwise logical AND of the lower 16 bits registers Dn and Dm, and stores the result in the lower 16 bits of register Dm. The upper 8 bits of register Dm will not change.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Always 0.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F3 : 00+Dn<<2+Dm

AND imm8, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	●	●

Operation

Dn&(x'FF0000' | imm8) → Dn

Performs a bitwise logical AND of zero-extends the 8-bit immediate value imm8 to 16-bits and the lower 16 bits of register Dn, and stores the result in the lower 16 bits of register Dn. The upper 8 bits of register Dn will not change.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Always 0.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 3
Cycles: 2

F5 : 00+Dn : imm8

The 8-bit immediate value imm8 will be zero-extended to 16 bits.

AND imm16, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	●	●

Operation

Dn&(x'FF0000' I imm16) → Dn

Performs a bitwise logical AND of the 16-bit immediate value imm16 and the lower 16 bits of register Dn, and stores the result in the lower 16 bits of register Dn. The upper 8 bits of register Dn will not change.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Always 0.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 4
Cycles: 2

F7 : 00+Dn : imm16-l : imm16-h

AND imm16, PSW

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

PSW&imm16 → PSW

Performs a bitwise logical AND of the 16-bit immediate value imm16 and the PSW, and stores the result in the PSW. The lower 8 bits will be stored in the PSW flags VX (bit 7), CX (bit 6), NX (bit 5), ZX (bit 4), VF (bit 3) , CF (bit 2), NF (bit 1), and ZF (bit 0). The PSW flags S1, S0, IE, and IM2~IM0 will also be changed.

Flag Changes

VX: Will be set to bit 7 of the result.
CX: Will be set to bit 6 of the result.
NX: Will be set to bit 5 of the result.
ZX: Will be set to bit 4 of the result.
VF: Will be set to bit 3 of the result.
CF: Will be set to bit 2 of the result.
NF: Will be set to bit 1 of the result.
ZF: Will be set to bit 0 of the result.

Size, Cycles, Codes

Bytes: 4
Cycles: 3

F7 : 10 : imm16-l : imm16-h

OR

Logical Calculation Instructions

OR Dn, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	●	●

Operation

Dm | (Dn&x'00FFFF') → Dm

Performs a bitwise logical OR of the lower 16 bits or registers Dm and Dn, and stores the result in the lower 16 bits of register Dm. The upper 8 bits of register Dm will not change.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Always 0.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F3 : 10 +Dn<< 2+Dm

OR imm8, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	●	●

Operation

Dn | imm8 → Dn

Performs a bitwise logical OR of zero-extends the 8-bit immediate value imm8 to 16-bits and the lower 16 bits of register Dn, and stores the result in the lower 16 bits of register Dn. The upper 8 bits of register Dn will not change.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Always 0.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 3
Cycles: 2

F5 : 08 +Dn : imm8

(!) The 8-bit immediate value imm8 will be zero-extended to 16 bits.

OR imm16, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	●	●

Operation

Dn I imm16 → Dn

Performs a bitwise logical OR of the lower 16 bits or registers Dn and the 16-bit immediate value imm16, and stores the result in the lower 16 bits of register Dn. The upper 8 bits of register Dn will not change.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Always 0.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 4
Cycles: 2

F7 : 40+Dn : imm16-l : imm16-h

OR imm16, PSW

VX	CX	NX	ZX	VF	CF	NF	ZF
●	●	●	●	●	●	●	●

Operation

PSW I imm16 → PSW

Performs a bitwise logical OR of the 16-bit immediate value imm16 and the PSW, and stores the result in the PSW. The lower 8 bits will be stored in the PSW flags VX (bit 7), CX (bit 6), NX (bit 5), ZX (bit 4), VF (bit 3), CF (bit 2), NF (bit 1), and ZF (bit 0). The PSW flags S1, S0, IE, and IM2~IM0 will also be changed.

Flag Changes

VX: Will be set to bit 7 of the result.
CX: Will be set to bit 6 of the result.
NX: Will be set to bit 5 of the result.
ZX: Will be set to bit 4 of the result.
VF: Will be set to bit 3 of the result.
CF: Will be set to bit 2 of the result.
NF: Will be set to bit 1 of the result.
ZF: Will be set to bit 0 of the result.

Size, Cycles, Codes

Bytes: 4
Cycles: 3

F7 : 14 : imm16-l : imm16-h

XOR Logical Calculation Instructions

XOR Dn, Dm

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	●	●

Operation

Dm ^ (x'00FFFF'&Dn) → Dm

Performs a bitwise logical XOR of the lower 16 bits or registers Dm and Dn, and stores the result in the lower 16 bits of register Dm. The upper 8 bits of register Dm will not change.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Always 0.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F3 :20 +Dn<<2+Dm

XOR imm16, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	●	●

Operation

Dn ^ imm16 → Dn

Performs a bitwise logical XOR of the lower 16 bits or registers Dn and the immediate value imm16, and stores the result in the lower 16 bits of register Dn. The upper 8 bits of register Dn will not change.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Always 0.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 4
Cycles: 2

F7 : 4C+Dn : imm16-l : imm16-h

NOT

Logical Calculation Instructions

NOT Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	●	●

Operation

Dn ^ x'00FFFF'→ Dn

Inverts each of the lower 16 bits in register Dn, and stores the result in the lower 16 bits of register Dn. The upper 8 bits of register Dn will not change.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Always 0.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

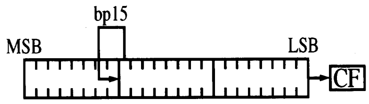
F3 : E4+Dn

ASR Logical Calculation Instructions

ASR Dn	VX	CX	NX	ZX	VF	CF	NF	ZF
	—	—	—	—	0	●	●	●

Operation

Dn.lsb → CF
Dn.bp → Dn.bp-1(bp15~1)
Dn.bp15 → Dn.bp15



Performs a 1-bit arithmetic shift right (towards the LSB) on the lower 16-bit value in register Dn, and stores the result in the lower 16 bits of register Dn. The value of bit 15 will not change. The LSB before the shift will be stored in the 16-bit data carry flag (CF). The upper 8 bits of register Dn will not change.

Flag Changes	Size, Cycles, Codes
VX , CX , NX , ZX : No change. VF: Always 0. CF: Set if the LSB of the beforeoperation is 1; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 2 Cycles: 2 F3 : 38+Dn

Use ADD Dn, Dm to perform a shift left of one bit. (See Section 2.19 in Chapter 3 "Use of Instructions.")

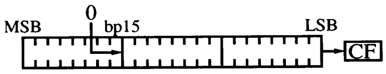
LSR Logical Calculation Instructions

LSR Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	●	0	●

Operation

Dn.lsb → CF
Dn.bp → Dn.bp-1(bp15~1)
0 → Dn.bp15



Performs a 1-bit logic shift right (towards the LSB) on the lower 16-bit value in register Dn, and stores in the register Dn. The value of bit 15 stores 0. The LSB from before the shift will be stored in the 16-bit data carry flag (CF). The upper 8 bits of register Dn will not change.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Set if the LSB of the beforeoperation is 1; reset otherwise.
NF: Always 0.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2
Cycles: 2

F3 : 3C+Dn

Use ADD Dn, Dm to perform a shift left of one bit. (See Section 2.19 in Chapter 3 "Use of Instructions.")

ROR

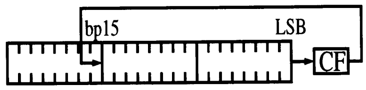
Logical Calculation Instructions

ROR Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	●	●	●

Operation

Dn.lsb → temp
Dn.bp → Dn.bp-1(bp15~1)
CF → Dn.bp15
temp → CF



Performs a 1-bit rotate right (towards the LSB) on the lower 16-bit value in register Dn and the carry flag (CF), and stores the result in the lower 16 bits of register Dn. The value of the 16-bit data carry flag (CF) before the shift will be stored in bit 15. The LSB before the shift will be stored in the carry flag (CF). The upper 8 bits of register Dn will not change.

Flag Changes	Size, Cycles, Codes
VX , CX , NX , ZX : No change. VF: Always 0. CF: Set if the LSB of the beforeoperation is 1; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 2 Cycles: 2 F3 : 34+Dn

ROL

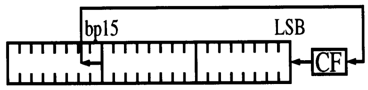
Logical Calculation Instructions

ROL Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	●	●	●

Operation

Dn.bp15 → temp
Dn.bp → Dn.bp+1(bp14~0)
CF → Dn.lsb
temp → CF



Performs a 1-bit rotate left (towards the MSB) on the lower 16-bit value in register Dn and the carry flag (CF), and stores the result in the lower 16 bits of register Dn. The value of the carry flag (CF) before the shift will be stored in bit 15. The LSB before the shift will be stored in the carry flag (CF). The upper 8 bits of register Dn will not change.

Flag Changes	Size, Cycles, Codes
VX , CX , NX , ZX : No change. VF: Always 0. CF: Set if the LSB of the beforeoperation is 1; reset otherwise. NF: Set if bit 15 of the result is 1; reset otherwise. ZF: Set if the lower 16 bits of the result are 0; reset otherwise.	Bytes: 2 Cycles: 2 F3 : 30+Dn

BTST

Bit Manipulation Instructions

BTST imm8, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	0	●

Operation

Dn&imm8 ··· PSW

Zero-extends the 8-bit immediate value imm8 to 24 bits, performs a logical AND with the value of register Dn, and reflects the result in the PSW flags.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Always 0.
NF: Always 0.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 3
Cycles: 2

F5 : 04 +Dn : imm8

The 8-bit immediate value imm8 will be zero-extended to 24 bits.

BTST imm16, Dn

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	●	●

Operation

Dn&imm16 ··· PSW

Zero-extends the 16-bit immediate value imm16 to 24 bits, performs a logical AND with the value of register Dn, and reflects the result in the PSW flags.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Always 0.
NF: Set if bit 15 of the result is 1; reset otherwise.
ZF: Set if the lower 16 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 4
Cycles: 2

F7 : 04+Dn : imm16-l : imm16-h

The16-bit immediate value imm16 will be zero-extended to 24 bits.

BSET

Bit Manipulation Instructions

BSET Dm, (An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	0	●

Operation

mem8(An)&Dm ··· PSW

mem8(An) | Dm → mem8(An)

Zero-extends the 1-byte contents of memory pointed to by register An to 24 bits, performs a logical AND with the value of register Dm, and reflects the result in the PSW flags. Also performs a logical OR of the zero-extended value and the value of register Dm, and stores the result in the lower 8 bits in the memory pointed to by register An. This instruction does not change the value of register Dm.

Flag Changes

VX , CX , NX , ZX : No change.
VF: Always 0.
CF: Always 0.
NF: Always 0.
ZF: Set if the lower 8 bits of the result are 0; reset otherwise.

Size, Cycles, Codes

Bytes: 2

Cycles: 5

F0 : 20 +An<<2+Dm

! When BSET instruction is executed, bus-line release request and interrupt request won't be accepted.

BCLR

Bit Manipulation Instructions

BCLR Dm, (An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	0	0	0	●

Operation

mem8(An)&Dm ··· PSW

mem8(An)&(~Dm) → mem8(An)

Zero-extends the 1-byte contents of memory pointed to by register An to 24 bits, performs a logical AND with the value of register Dm, and reflects the result in the PSW flags. Also performs a logical OR of the zero-extended value and the inverted value of register Dm, and stores the result in the lower 8 bits in the memory pointed to by register An. This instruction does not change the value of register Dm.

Flag Changes	Size, Cycles, Codes
VX , CX , NX , ZX : No change. VF: Always 0. CF: Always 0. NF: Always 0. ZF: Set if the lower 8 bits of the first result are 0; reset otherwise.	Bytes: 2 Cycles: 5 F0 : 30 +An<<2+Dm

When BCLR instruction is executed, bus-line release request and interrupt request won't be accepted.

Bcc Branch Instructions

BEQ label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF ZF = 1, then PC+2+d8(label) → PC
IF ZF = 0, then PC+2 → PC

If ZF is 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If ZF is 0, then execution will continue with the instruction following the BEQ instruction. For example, the BEQ instruction will branch when the previous CMP instruction encounters a 16-bit source equal to a 16-bit destination.

Flag Changes

No change.

Size, Cycles, Codes

Bytes: 2
Cycles: 2 (branch)
Cycle: 1 (non-branch)

E8 : d8

BNE label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF ZF = 0, then PC+2+d8(label) → PC
IF ZF = 1, then PC+2 → PC

If ZF is 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If ZF is 1, then execution will continue with the instruction following the BNE instruction. For example, the BNE instruction will branch when the previous CMP instruction encounters a 16-bit source not equal to a 16-bit destination.

Flag Changes

No change.

Size, Cycles, Codes

Bytes: 2
Cycles: 2 (branch)
Cycle: 1 (non-branch)

E9 : d8

BLT label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF (VF ^ NF) = 1, then PC+2+d8(label) → PC
IF (VF ^ NF) = 0, then PC+2 → PC

If VF is 0 and NF is 1, or if VF is 1 and NF is 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If both VF and NF are 0, or if both VF and NF are 1, then execution will continue with the instruction following the BLT instruction. For example, the BLT instruction will branch when the previous CMP instruction encounters src>dest as a signed 16-bit value.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 2 Cycles: 2 (branch) Cycle: 1 (non-branch) E0 : d8

BLE label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF ((VF ^ NF) I ZF) = 1, then PC+2+d8(label) → PC
IF ((VF ^ NF) I ZF) = 0, then PC+2 → PC

If VF is 0 and NF is 1, or if VF is 1 and NF is 0, or if ZF is 1,then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If both VF, NF and ZF are 0, both VF, NF and ZF are 1 and NF is 0, or if ZF is 1, then execution will continue with the instruction following the BLE instruction. For example, the BLE instruction will branch when the previous CMP instruction encounters src≧dest as a signed 16-bit value.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 2 Cycles: 2 (branch) Cycle: 1 (non-branch) E3 : d8

BGE label	VX	CX	NX	ZX	VF	CF	NF	ZF
	—	—	—	—	—	—	—	—

Operation

IF (VF ^ NF) = 0, then PC+2+d8(label) → PC
IF (VF ^ NF) = 1, then PC+2 → PC

If VF and NF are 0, or if VF and NF are 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If VF is 0 and NF is 1, or if VF is 1 and NF is 0, then execution will continue with the instruction following the BGE instruction. For example, the BGE instruction will branch when the previous CMP instruction encounters src≦dest as a signed 16-bit value.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 2 Cycles: 2 (branch) Cycle: 1 (non-branch) E2 : d8

BGT label	VX	CX	NX	ZX	VF	CF	NF	ZF
	—	—	—	—	—	—	—	—

Operation

IF ((VF ^ NF) I ZF) = 0, then PC+2+d8(label) → PC
IF ((VF ^ NF) I ZF) = 1, then PC+2 → PC

If both VF, NF and ZF are 0, or if both VF,NF and ZF are 1 and if ZF is 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If VF is 0 and NF is 1, or if VF is 1 and NF is 0, or if ZF is 1 then execution will continue with the instruction following the BGT instruction. For example, the BGT instruction will branch when the previous CMP instruction encounters src<dest as a signed 16-bit value.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 2 Cycles: 2 (branch) Cycle: 1 (non-branch) E1 : d8

BCS label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF CF = 1, then PC+2+d8(label) → PC
IF CF = 0, then PC+2 → PC

If CF is 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If CF is 0, then execution will continue with the instruction following the BCS instruction. For example, the BCS instruction will branch when the previous CMP instruction encounters src>dest as a un-signed 16-bit value.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 2 Cycles: 2 (branch) Cycle: 1 (non-branch) E4 : d8

BLS label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF (CF I ZF) = 1, then PC+2+d8(label) → PC
IF (CF I ZF) = 0, then PC+2 → PC

If CF is 1, or ZF is 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If both CF and ZF are 0, then execution will continue with the instruction following the BLS instruction. For example, the BLS instruction will branch when the previous CMP instruction encounters src≧dest as a signed 16-bit value.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 2 Cycles: 2 (branch) Cycle: 1 (non-branch) E7 : d8

BCC label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF CF = 0, then PC+2+d8(label) → PC
IF CF = 1, then PC+2 → PC

If CF is 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If CF is 1, then execution will continue with the instruction following the BCC instruction. For example, the BCC instruction will branch when the previous CMP instruction encounters src≦dest as a un-signed 16-bit value.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 2 Cycles: 2 (branch) Cycle: 1 (non-branch) E6 : d8

BHI label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF (CF I ZF) = 0, then PC+2+d8(label) → PC
IF (CF I ZF) = 1, then PC+2 → PC

If both CF and ZF are 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If CF is 1, or if ZF is 1, then execution will continue with the instruction following the BHI instruction.For example, the BHI instruction will branch when the previous CMP instruction encounters src<dest as a un-signed 16-bit value.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 2 Cycles: 2 (branch) Cycle: 1 (non-branch) E5 : d8

BVC label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF VF = 0, then PC+3+d8(label) → PC
IF VF = 1, then PC+3 → PC

If VF is 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If VF is 1, then execution will continue with the instruction following the BVC instruction.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : FC : d8

BVS label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF VF = 1, then PC+3+d8(label) → PC
IF VF = 0, then PC+3 → PC

If VF is 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If VF is 0, then execution will continue with the instruction following the BVS instruction.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : FD : d8

BNC label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF NF = 0, then PC+3+d8(label) → PC
IF NF = 1, then PC+3 → PC

If NF is 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If NF is 1, then execution will continue with the instruction following the BNC instruction.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : FE : d8

BNS label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF NF = 1, then PC+3+d8(label) → PC
if NF = 0, then PC+3 → PC

If NF is 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If NF is 0, then execution will continue with the instruction following the BNS instruction.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 :FF : d8

BRA label	VX	CX	NX	ZX	VF	CF	NF	ZF
	—	—	—	—	—	—	—	—

Operation

PC+2+d8(label) → PC

Branches unconditionally to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after.

Flag Changes	Size, Cycles, Codes
No change.	Bytes: 2 Cycles: 2 EA : d8

Bccx Branch Instructions

BEQX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF ZX = 1, then PC+3+d8(label) → PC
IF ZX = 0, then PC+3 → PC

If ZX is 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If ZX is 0, then execution will continue with the instruction following the BEQX instruction. For example, the BEQX instruction will branch when the previous CMP instruction encounters a 24-bit source equal to a 24-bit destination.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : E8 : d8

BNEX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF ZX = 0, then PC+3+d8(label) → PC
IF ZX = 1, then PC+3 → PC

If ZX is 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If ZX is 1, then execution will continue with the instruction following the BNEX instruction. For example, the BNEX instruction will branch when the previous CMP instruction encounters a 24-bit source not equal to a 24-bit destination.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : E9 : d8

BLTX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF (VX ^ NX) = 1, then PC+3+d8(label) → PC
IF (VX ^ NX) = 0, then PC+3 → PC

If VX is 0 and NX is 1, or if VX is 1 and NX is 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If both VX and NX are 0, or if both are 1, then execution will continue with the instruction following the BLTX instruction. For example, the BLTX instruction will branch when the previous CMP instruction encounters src>dest as a signed 24-bit value.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : E0 : d8

BLEX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF ((VX ^ NX) I ZX) = 1, then PC+3+d8(label) → PC
IF ((VX ^ NX) I ZX) = 0, then PC+3 → PC

If VX is 0 and NX is 1, or if VX is 1 and NX is 0, or if ZX is 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If both VX,NX and ZX are 0, or if both VX and NX are 1 and ZX is 0, then execution will continue with the instruction following the BLEX instruction. For example, the BLEX instruction will branch when the previous CMP instruction encounters src≧dest as a signed 24-bit value.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : E3 : d8

BGEX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF (VX ^ NX) = 0, then PC+3+d8(label) → PC
IF (VX ^ NX) = 1, then PC+3 → PC

If both VX and NX are 0, or if both VX and NX are 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If VX is 0 and NX is 1, or if VX is 1 and NX is 0, then execution will continue with the instruction following the BGEX instruction.For example, the BGEX instruction will branch when the previous CMP instruction encounters src≦dest as a signed 24-bit value.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : E2 : d8

BGTX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF ((VX ^ NX) I ZX) = 0, then PC+3+d8(label) → PC
IF ((VX ^ NX) I ZX) = 1, then PC+3 → PC

If both VX,NX and ZXare 0, or if both VX and NX are 1 and ZX is 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If VX is 0 and NX is 1, or if VX is 1 and NX is 0, or if ZX is 1,then execution will continue with the instruction following the BGTX instruction. For example, the BGTX instruction will branch when the previous CMP instruction encounters src<dest as a signed 24-bit value.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : E1 : d8

BCSX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF CX = 1, then PC+3+d8(label) → PC
IF CX = 0, then PC+3 → PC

If CX is 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If CX is 0, then execution will continue with the instruction following the BCSX instruction.For example, the BCSX instruction will branch when the previous CMP instruction encounters src>dest as a un-signed 24-bit value.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : E4 : d8

BLSX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF (CX | ZX) = 1, then PC+3+d8(label) → PC
IF (CX | ZX) = 0, then PC+3 → PC

If CX is 1 or ZX is 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If both CX and ZX are 0, then execution will continue with the instruction following the BLSX instruction.For example, the BLSX instruction will branch when the previous CMP instruction encounters src≧dest as a un-signed 24-bit value.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : E7 : d8

BCCX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF CX = 0, then PC+3+d8(label) → PC
IF CX = 1, then PC+3 → PC

If CX is 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If CX is 1, then execution will continue with the instruction following the BCCX instruction.For example, the BCCX instruction will branch when the previous CMP instruction encounters src≦dest as a un-signed 24-bit value.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : E6 : d8

BHIX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF (CX I ZX) = 0, then PC+3+d8(label) → PC
IF (CX I ZX) = 1, then PC+3 → PC

If both CX and ZX are 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If CX or ZX is 1, then execution will continue with the instruction following the BHIX instruction.For example, the BHIX instruction will branch when the previous CMP instruction encounters src<dest as a un-signed 24-bit value.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : E5 : d8

BVCX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF VX = 0, then PC+3+d8(label) → PC
IF VX = 1, then PC+3 → PC

If VX is 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If VX is 1, then execution will continue with the instruction following the BVCX instruction.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : EC : d8

BVSX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF VX = 1, then PC+3+d8(label) → PC
IF VX = 0, then PC+3 → PC

If VX is 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If VX is 0, then execution will continue with the instruction following the BVSX instruction.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : ED : d8

BNCX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF NX = 0, then PC+3+d8(label) → PC
IF NX = 1, then PC+3 → PC

If NX is 0, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If NX is 1, then execution will continue with the instruction following the BNCX instruction.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : EE : d8

BNSX label

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

IF NX = 1, then PC+3+d8(label) → PC
IF NX = 0, then PC+3 → PC

If NX is 1, then branches to the address indicated by "label". The branch range is from 128 bytes before the first address of the next instruction to 127 bytes after. If NX is 0, then execution will continue with the instruction following the BNSX instruction.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 3 (branch) Cycles: 2 (non-branch) F5 : EF : d8

JMP

Branch Instructions

JMP label16

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

PC+3+d16(label16) → PC

Branches unconditionally to the address indicated by "label". The branch range is from 32768 bytes before the first address of the next instruction to 32767 bytes after.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 3 Cycles: 2 FC : d16-l : d16-h

JMP label24

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

PC+5+d24(label24) → PC

Branches unconditionally to the address indicated by "label". The branch range is from 8388608 bytes before the first address of the next instruction to 8388607 bytes after.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 5 Cycles: 4 F4 : E0 : d24-l : d24-m : d24-h

(!) The assembler will determine whether d16 or d24 is in optimazation processing.

JMP (An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation	
	An → PC Branches unconditionally to the address indicated by register An.

Flag Changes	Size, Cycles, Codes
No changes.	Bytes: 2 Cycles: 3 F0 : An<<2

JSR

Branch Instructions

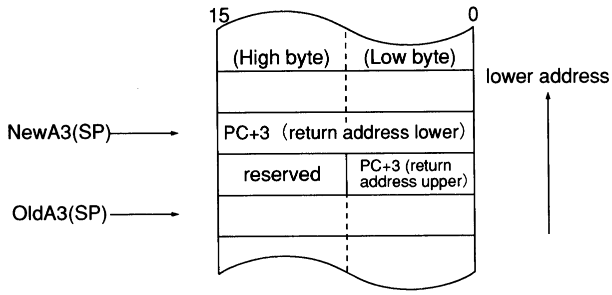
JSR label16

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

A3-4 → A3
PC+3 → mem24(A3)
PC+3+d16(label16) → PC

Calls the subroutine at the address indicated by "label16". The subroutine call range is from 32768 bytes before the first address of the next instruction to 32767 bytes after. The stack pointer value will be subtracted by 4, and the address of the next instruction after the JSR instruction (the return address) will then be stored at the stack pointer. After the JSR instruction executes, the stack will be as shown at right.



Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 3
Cycles: 4

FD : d16-l : d16-h

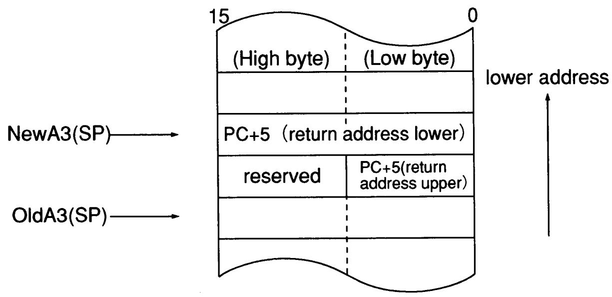
JSR label24

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

A3-4 → A3
PC+5 → mem24(A3)
PC+5+d24(label24) → PC

Calls the subroutine at the address indicated by "label24". The stack pointer value will be subtracted by 4, and the address of the next instruction after the JSR instruction (the return address) will then be stored at the stack pointer. After the JSR instruction executes, the stack will be as shown at right.



Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 5
Cycles: 5

F4 : E1 : d24-l : d24-m : d24-h

The assembler will determine whether d16 or d24 is in optimazation processing.

JSR (An)

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

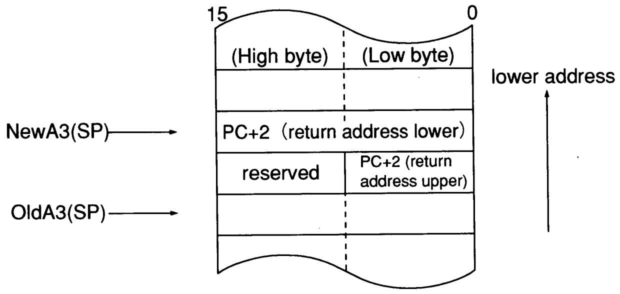
Operation

A3-4 → A3

PC+2 → mem24(A3)

An → PC

Calls the subroutine at the address pointed to by register An. The stack pointer value will be subtracted by 4, and the address of the next instruction after the JSR instruction (the return address) will then be stored at the stack pointer. After the JSR instruction executes, the stack will be as shown at right.



Flag Changes

No changes.

Size, Cycles, Codes

Bytes: 2

Cycles: 5

F0 : 01+An<<2

NOP

Branch Instructions

NOP

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

PC+1 → PC

Proceeds to the next instruction without performing any operation.

Flag Changes	Size, Cycles, Codes
No changes.	Byte: 1 Cycle: 1 F6

RTS Branch Instructions

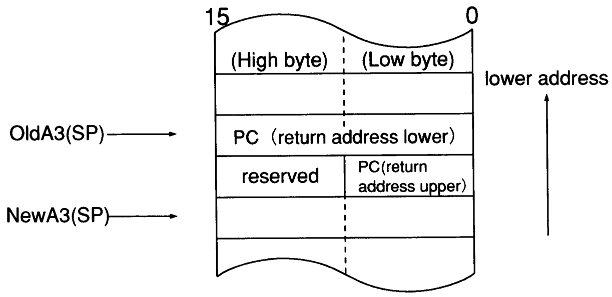
RTS

VX	CX	NX	ZX	VF	CF	NF	ZF
—	—	—	—	—	—	—	—

Operation

mem24(A3) → PC
A3+4 → A3

Returns from a subroutine to the original program. The address of the next instruction to execute will be popped from the stack into the PC, and 4 will be added to the stack pointer. After the RTS instruction executes, the stack will be as shown at right.



Flag Changes

No changes.

Size, Cycles, Codes

Byte: 1
Cycles: 5

FE

RTI

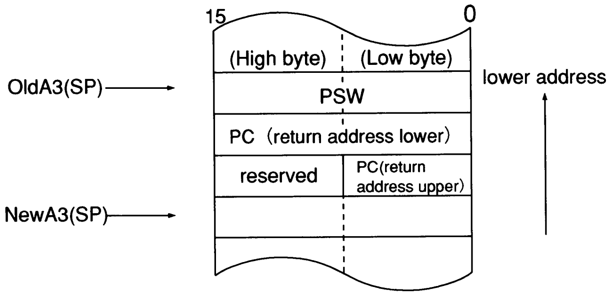
Branch Instructions

RTI	VX	CX	NX	ZX	VF	CF	NF	ZF
	●	●	●	●	●	●	●	●

Operation

mem16(A3) → PSW
mem24(A3+2) → PC
A3+6 → A3

Returns from an interrupt process to the program that was executing before the interrupt was received. The PSW before the interrupt will be popped from the stack into the PSW, and the address of the next instruction to execute will be popped from the stack into the PC, and 6 will be added to the stack pointer. After the RTI instruction executes, the stack will be as shown at right.



Flag Changes	Size, Cycles, Codes
VX : Restored to its pre-interrupt status. CX : Restored to its pre-interrupt status. NX : Restored to its pre-interrupt status. ZX : Restored to its pre-interrupt status. VF : Restored to its pre-interrupt status. CF : Restored to its pre-interrupt status. NF : Restored to its pre-interrupt status. ZF : Restored to its pre-interrupt status.	Byte: 1 Cycles: 6 EB

Use Of Instructions

3

Notes Regarding Use Of Instructions

Chapter 3, "Use Of Instructions," collects notes, precautions, and know-how for instruction selection that the user must know when using instructions in the MN102L series. This information can be divided into two categories.

(1) Minimum knowledge Required
(2) Programming examples: general, speed optimization, size optimization

Each item of information consists of the following.

【Contents】
This describes what the program does. Also explains such conditions as memory model for use.

【Category】

Categories are shown by the following icons.

Precaution
This is a precaution that the user absolutely should know.

Speed optimization
This is a technique for increasing code speed.

Size optimization
This is a technique for decreasing code size.

General information

【Example】
These are actual assembly language program examples. The code size and cycle count are shown after each instruction as shown below.

instruction ;　　　　(code size, cycle count)

- The execution cycle counts shown in the code examples indicate the minimum cycles for each instruction, so actual counts may increase depending on conditions for use. Note that these cycle counts may differ from those listed in the appendix.
- Cycle counts will also differ for single-chip mode, memory expansion mode, and processor mode. Specific cycle counts should be measured with an emulator.

---

1 Minimum Knowledge Required

1-1 Word Accesses To Odd Addresses

Precaution

【Contents】

Word accesses and pointer accesses to odd addresses cannot be performed. The stack pointer (A3) also should not hold an odd address. If the value of A3 is odd, then JSR, RTS, and similar instructions that use the stack area will not operate correctly. By restricting word accesses and pointer accesses to odd addresses, the MN102 designs possibly increase RAM size requirements but are able to make faster accesses.

【Example】

① Word Accesses
```
MOV     (x'f001') , D0     ;Odd address not allowed.
        ↓   (Code as follows.)
MOV     (x'f000') ,D0
```

② Stack Pointer Operation
```
ADD     1,A3               ;Odd address not allowed.
        ↓   (Code as follows.)
ADD     2,A3
```

③ Stack Pointer Operation
```
ADD     -1,A3              ;Odd address not allowed.
        ↓   (Code as follows.)
ADD     -2,A3
```

1 - 2 Increment/Decrement Of Address Registers

Precaution

【Contents】

Address registers are incremented and decremented by ADD and SUB instructions, but the PSW flags will change according to the results. To use PSW flags set according to the results of a subroutine after returning from the subroutine, use the ADDNF instruction, which does not change PSW flags, to adjust the stack pointer when returning from the subroutine.

【Example】

① Increment/Decrement Of Address Registers

```
ADD     -2,A0
BEQ     error
```

② Stack Pointer Operation Without Changing PSW Flags

```
MOV     (A3),A1
MOV     (4,A3),A2
MOVX    (8,A3),D2
MOVX    (12,A3),D3
ADDNF   16,A3
RTS
```

1 - 3 24-Bit Pointer Operations

【Contents】

Pointer data is handled as 24-bit data. Register saves/restores for subroutines are always processed as pointer data (24 bits). The data register Dn makes use of MOVX instructions.

【Example】

Preprocessing and postprocessing of subroutine kyo(int column, char *sou, char *dist).

```
            .SECTION        _TEXT, CODE, ALIGN=2
;--------------------------------------------------------------------------
BYTESIZ     .equ    1                               ;Byte size
INTSIZ      .equ    2                               ;Integer size
PTRSIZ      .equ    4                               ;Pointer size (linear version)
LONGSIZ     .equ    4                               ;Long size
;--------------------------------------------------------------------------
push_D2     .assign 0
push_D3     .assign push_D2  + PTRSIZ
push_A1     .assign push_D3  + PTRSIZ
push_A2     .assign push_A1  + PTRSIZ
;--------------------------------------------------------------------------

;-----------------------------------------------
        Stack frame
;-----------------------------------------------
pushwk      .assign 0                               ;Storage area for D2, D3, A1, and A2
;-----------------------------------------------Subroutine work area follows
work_size   .assign pushwk      + PTRSIZ*4
;-----------------------------------------------Parameters
colmun      .assign work_size   + PTRSIZ*2          ;contcolumn
sou         .assign colmun      + INTSIZ            ;wromadr
dist        .assign sou         + PTRSIZ            ;wramadr
;--------------------------------------------------------------------------
            .export _kyo
_kyo:
            DEFINE      work_size=16
            ADD         -work_size, A3  ;Preprocessing
            MOV         A1,(push_A1,A3) ┐
            MOV         A2,(push_A2,A3) │Register storage
            MOVX        D2,(push_D2,A3) │
            MOVX        D3,(push_D3,A3) ┘

            MOV         (colmun,A3),D2 ;get column ┐
            MOV         (sou,A3),A0                │Parameters
            MOV         (dist,A3),A1               ┘
                        .
                        .
                        .
 Norm_end:

            SUB         D0,D0           ;Postprocessing
            MOV         (push_A1,A3),A1
            MOV         (push_A2,A3),A2
            MOVX        (push_D2,A3),D2
            MOVX        (push_D3,A3),D3
            ADD         work_size,A3
            RTS
```

2 Programming Examples: General, Speed Optimization, Size Optimization

2 - 1 Register Initialization

Precaution

【Contents】

All registers are undefined after a reset start. Register initialization must be performed first. Especially do not forget to set the stack pointer A3.

【Example】

Clear all registers to 0, and set the stack pointer to label "stack."

```
SUB   D0,D0          ;Clear D0
MOV   D0,D1
MOV   D0,D2
MOV   D0,D3
MOV   D0,A0
MOV   D0,A1
MOV   D0,A2
MOV   a(stack),A3    ;Set A3 to initial value of stack pointer.
```

2 - 2 I/O Access

Precaution

【Contents】

All MOV instructions, MOVB instructions, and MOVBU instructions can be used for input/output operations.

【Example】

① Output from Port 1

```
MOVB    D0, (p1out)
```

② Input from Port 2

```
MOVBU   (p2in), D0
```

2 - 3 Storing Immediate Data In Memory

Speed optimization　Size optimization

【Contents】

To store an immediate value in memory, first store the immediate value in a register and then move it to memory with a MOV instruction. When storing a 24-bit pointer immediate value in memory, code size and speed is better if the move is through an address register than if it is through a data register.

【Example】

```
① Store an immediate value in memory.
MOV     x'1234',D0        ; (3 bytes, 1 cycle)
MOV     D0,(A1)           ; (1 byte, 1 cycle)
                   Total    (4 bytes, 2 cycles)

② Store a pointer immediate value in memory.
MOV     x'123456',D0      (5 bytes, 3 cycles)
MOVX    D0,(0,A1)         (3 bytes, 3 cycles)
                   Total  (8 bytes, 6 cycles)
   ↓ (Moving through an address register gives faster, smaller code.)
MOV     x'123456',A0      (5 bytes, 3 cycles)
MOV     A0,(A1)           (2 bytes, 2 cycles)
                   Total  (7 bytes, 5 cycles)
```

Speed optimization

Size optimization

2 - 4 Data Move From Memory To Memory

【Contents】

Data moves from memory to memory are performed through a register. When moving 24-bit pointer data, code size and speed is better if the move is through an address register than if it is through a data register.

【Example】

```
① Data Moves
MOV     (x'f000'),D0        ; (3 bytes, 1 cycle)
MOV     D0,(x'f002')        ; (3 bytes, 1 cycle)
                      Total   (6 bytes, 2 cycles)
MOV     x'f000',A0          ; (3 bytes, 1 cycle)
MOV     (A0),D0             ; (1 byte, 1 cycle)
MOV     D0,(2,A0)           ; (2 bytes, 1 cycle)
                      Total   (6 bytes, 3 cycles)

② Pointer Data Moves
MOV     (x'f000'),A0        ; (4 bytes, 3 cycles)
MOV     A0,(x'f004')        ; (4 bytes, 3 cycles)
                      Total   (8 bytes, 6 cycles)
MOV     x'f000',A1          ; (3 bytes, 1 cycle)
MOV     (A1),A0             ; (2 bytes, 2 cycles)
MOV     A0,(4,A1)           ; (2 bytes, 2 cycles)
                      Total   (7 bytes, 5 cycles)
MOV     x'f000,A1           ; (3 bytes, 1 cycle)
MOVX    (0,A1),D0           ; (3 bytes, 3 cycles)
MOVX    D0,(4,A1)           ; (3 bytes, 3 cycles)
                      Total   (9 bytes, 7 cycles)
```

2 - 5 Repeated Access To The Same Memory

Size optimization

【Contents】

When performing repeated accesses to the same memory, register indirect addressing will result in smaller code size than absolute addressing.

【Example】

```
MOV     (x'f000'),D0      ; (3 bytes, 1 cycle)
ADD     1,D0              ; (2 bytes, 1 cycle)
MOV     D0,(x'f000')      ; (3 bytes, 1 cycle)
                    Total  (8 bytes, 3 cycles)

        ↓ (Register indirect addressing gives smaller code size.)

MOV     x'f000',A0        ; (3 bytes, 1 cycle)
MOV     (A0),D0           ; (1 byte, 1 cycle)
ADD     1,D0              ; (2 bytes, 1 cycle)
MOV     D0,(A0)           ; (1 byte, 1 cycle)
                    Total  (7 bytes, 4 cycles)
```

2 - 6 Byte Access And Word Access

Speed optimization　Size optimization

【Contents】

Word accesses result in better code speed and size than byte accesses.

【Example】

```
①
MOVB    (A3),D0           ;(2 bytes, 2 cycles)
        ↓ (Word accesses give smaller, faster code.)
MOV     (A3),D0           ;(1 byte, 1 cycle)

②
MOVB    (x'10',A0),D0     ;(3 bytes, 2 cycles)
        ↓ (Word accesses give smaller, faster code.)
MOV     (x'10',A0),D0     ;(2 bytes, 1 cycle)

③
MOVBU   (x'10',A0),D0     ;(3 bytes, 2 cycles)
        ↓ (Word accesses give smaller, faster code.)
MOV     (x'10',A0),D0     ;(2 bytes, 1 cycle)
```

2 - 7 Byte Move From Memory To Register

Speed optimization Size optimization

【Contents】

When moving bytes from memory to registers, zero extending results in better code size and speed.

【Example】

```
MOVB    (A3),D0                  ; (2 bytes, 2 cycles)
        ↓ (Zero extending gives smaller, faster code.)
MOVBU   (A3),D0                  ; (1 byte, 1 cycle)
```

2 - 8 Zero-Check Of Memory

Speed optimization Size optimization

【Contents】

Flags do not change when moving data from memory to registers. To check if a memory value is zero, another instruction that does change the flags needs to be used. In such cases CMP instructions will be faster than AND instructions. Furthermore, if it does not matter whether or not the moved data in the register is valid, then ADD instructions will result in smaller code size. CF and ZF will reflect the results.

【Example】

```
① Register data must be valid.
MOV     (x'f000'),D0       ; (3 bytes, 1 cycle)
AND     D0,D0              ; (2 bytes, 2 cycles)
                     Total   (5 bytes, 3 cycles)

        ↓ (CMP instruction is faster.)

MOV     (x'f000'),D0       ; (3 bytes, 1 cycle)
CMP     0,D0               ; (2 bytes, 1 cycle)
                     Total   (5 bytes, 2 cycles)

② Register data may be invalid.
MOV     (x'f000'),D0       ; (3 bytes, 1 cycle)
CMP     0,D0               ; (2 bytes, 1 cycle)
                     Total   (5 bytes, 2 cycles)

        ↓ (ADD instruction gives smaller code.)

MOV     (x'f000'),D0       ; (3 bytes, 1 cycle)
ADD     D0,D0              ; (1 byte, 1 cycle)
                     Total   (4 bytes, 2 cycles)
```

2 - 9 Block Move

General information

【Content】

Block moves are performed by a sequence of several instructions. There are many examples trading off speed, size, and numbers of registers used, but use ① below as a typical example.

【Example】

Move 16 bytes from x'f000'~x'f00f ' to x'f100'~x'f10f'.

① Use two data registers and two address registers (D0, D1, A0, A1)

```
      MOV x'f000',A0     ; (3 bytes, 1 cycle)
      MOV x'f100',A1     ; (3 bytes, 1 cycle)
      MOV x'8',D1        ; (2 bytes, 1 cycle)     Sub total (8 bytes, 3 cycles)
loop: MOV (A0),D0        ; (1 byte, 1 cycle)
      MOV D0,(A1)        ; (1 byte, 1 cycle)
      ADD2,A0            ; (2 bytes, 1 cycle)
      ADD2,A1            ; (2 bytes, 1 cycle)
      ADD-1,D1           ; (2 bytes, 1 cycle)
      BNE loop           ; (2 bytes, 2 /1 cycle)  Sub total (10 bytes,7/6 cycles)
                                                  Total (18 bytes, 59 cycles)
```

② Use three data registers and one address register (D0, D1, D2, A0)

```
      MOV x'f000',A0     ; (3 bytes, 1 cycle)
      MOV x'100',D2      ; (3 bytes, 1 cycle)
      MOV x'8',D1        ; (2 bytes, 1 cycle)     Sub total (8 bytes, 3 cycles)
loop: MOV (A0),D0        ; (1 byte, 1 cycle)
      MOV D0,(D2,A0)     ; (2 bytes, 2 cycles)
      ADD2,A0            ; (2 bytes, 1 cycle)
      ADD-1,D1           ; (2 bytes, 1 cycle)
      BNE loop           ; (2 bytes, 2/1 cycle)   Sub total (9 bytes, 7/6 cycles)
                                                  Total (17 bytes, 59 cycles)
```

③ Use two data registers and one address register (D0, D1, A0)

```
      MOV x'f000',A0     ; (3 bytes, 1 cycle)
      MOV x'8',D1        ; (2 bytes, 1 cycle)     Sub total (5 bytes, 2 cycles)
loop: MOV (A0),D0        ; (1 byte, 1 cycle)
      MOV D0,(x'100',A0) ; (4 bytes, 3 cycles)
      ADD2,A0            ; (2 bytes, 1 cycle)
      ADD-1,D1           ; (2 bytes, 1 cycle)
      BNE loop           ; (2 bytes, 2/1 cycle)   Sub total (11 bytes,8/7 cycles)
                                                  Total (16 bytes, 65 cycles)
```

④ Use two data registers and two address registers (D0, D1, A0, A1)

```
      MOV x'f000',A0     ; (3 bytes, 1 cycle)
      MOV x'f100',A1     ; (3 bytes, 1 cycle)
      MOV x'e',D1        ; (2 bytes, 1 cycle)     Sub total (8 bytes, 3 cycles)
loop: MOV (D1,A0),D0     ; (2 bytes, 2 cycle)
      MOV D0,(D1,A1)     ; (2 bytes, 2 cycles)
      ADD-2,D1           ; (2 bytes, 1 cycle)
      BGE loop           ; (2 bytes, 2/1 cycle)   Sub total (8 bytes, 7/6 cycles)
                                                  Total (16 bytes, 59 cycles)
```

2-10 PUSH・POP

Size optimization

【Contents】

Push and pop operations are performed by sequences of several instructions. By increasing the number of registers used to perform push and pop, example ② shows smaller code size.

【Example】

```
①
ADD     -4,A3                        ; (2 bytes, 1 cycle)
MOVX    D0,(A3)      ;push D0          (3 bytes, 3 cycles)
ADD     -4,A3                        ; (2 bytes, 1 cycle)
MOVX    D1,(A3)      ;push D1          (3 bytes, 3 cycles)
        :                      ; Total (10bytes, 8 cycles)
        :
MOVX    (A3),D1      ;pop D1           (3 bytes, 3 cycles)
ADD     4,A3                         ; (2 bytes, 1 cycle)
MOVX    (A3),D0      ;pop D0           (3 bytes, 3 cycles)
ADD     4,A3                         ; (2 bytes, 1 cycle)
                               ; Total (10 bytes, 8 cycles)

②
ADD     -8,A3                        ; (2 bytes, 1 cycle)
MOVX    D0,(4,A3)    ;push D0          (3 bytes, 3 cycles)
MOVX    D1,(A3)      ;push D1          (3 bytes, 3 cycles)
        :                      ; Total (8 bytes, 7 cycles)
        :
MOVX    (A3),D1      ;pop D1           (3 bytes, 3 cycles)
MOVX    (4,A3),D0    ;pop D0           (3 bytes, 3 cycles)
ADD     8,A3                         ; (2 bytes, 1 cycle)
                               ; Total (8 bytes, 7 cycles)
```

2-1 1 Push/Pop Of PSW

Precaution

【Contents】

When pushing and popping the PSW, the PSW flags will change if the stack pointer value is changed using an ADD instruction.

【Example】

```
① PUSH
ADD     -10,A3          ;PSW is changed.
MOVX    D0,(6,A3)
MOV     PSW,D0          ;Push the changed PSW.
MOV     D0,(4,A3)
MOVX    D1,(A3)

        ↓ (Code as follows.)

ADDNF   -10,A3          ;PSW is not changed.
MOVX    D0,(6,A3)
MOV     PSW,D0          ;Move PSW to D0.
MOV     D0,(4,A3)       ;Push PSW.
MOVX    D1,(A3)

② POP
MOVX    (A3),D1
MOV     (4,A3),D0
MOV     D0,PSW          ;Pop PSW.
MOVX    (6,A3),D0
ADD     10,A3           ;Poping PSW is changed.
        ↓ (Code as follows.)

MOVX    (A3),D1
MOV     (4,A3),D0       ;Move PSW to D0.
MOV     D0,PSW          ;Pop PSW.
MOVX    (6,A3),D0
ADDNF   10,A3
```

2-12 Zero-Clear Of Registers

Size optimization

【Contents】

To zero-clear registers, use the SUB instruction for smaller code size.

【Example】

```
MOV     0,D0                ; (2 bytes, 1 cycle)
        ↓ (SUB instruction gives smaller code size.)
SUB     D0,D0               ; (1 byte, 1 cycle)
```

2-13 Calculations With Memory Values

General information

【Contents】

To calculate with values in memory, the memory values must be moved to registers for the calculations.

【Example】

Add the values in memory at addresses x'f000'~x'f003' to addresses x'f010' ~x'f013' 32-bit addition).

```
MOV     x'f000',A0
MOV     x'f010',A1
MOV     (A0),D0
MOV     (A1),D1
ADD     D1,D0
MOV     D0,(A0)
MOV     (x'2',A0),D0
MOV     (x'2',A1),D1
ADDC    D1,D0
MOV     D0,(x'2',A0)
```

2-1 4 Bit Set

Size optimization

【Contents】

Normally OR instructions are used to set bits. For byte accesses the BSET instruction can be used in addition to an OR instruction.

Use the BSET instruction to prohibit reception of bus release requests and interrupts during read/modify/writes.

【Example】

① Word Access
```
MOV     (x'fc20'),D0     ; (3 bytes, 1 cycle)
OR      4,D0             ; (3 bytes, 2 cycles)
MOV     D0,(x'fc20')     ; (3 bytes, 1 cycle)
                   Total   (9 bytes, 4 cycles)
```

↓ (The example below gives smaller code size.)

```
MOV     x'fc20',A0       ; (3 bytes, 1 cycle)
MOV     (A0),D0          ; (1 byte, 1 cycle)
OR      4,D0             ; (3 bytes, 2 cycles)
MOV     D0,(A0)          ; (1 byte, 1 cycle)
                   Total   (8 bytes, 5 cycles)
```

② Using OR Instruction For Byte Access
```
MOVBU   (x'fc20'),D0     ; (3 bytes, 1 cycle)
OR      4,D0             ; (3 bytes, 2 cycles)
MOVB    D0,(x'fc20')     ; (3 bytes, 1 cycle)
                   Total   (9 bytes, 4 cycles)
```

↓ (The example below gives smaller code size.)

```
MOV     x'fc20',A0       ; (3 bytes, 1 cycle)
MOVBU   (A0),D0          ; (1 byte, 1 cycle)
OR      4,D0             ; (3 bytes, 2 cycles)
MOVB    D0,(A0)          ; (1 byte, 1 cycle)
                   Total   (8 bytes, 5 cycles)
```

③ Using BSET Instruction For Byte Access
```
MOV     x'fc20',A0       ; (3 bytes, 1 cycle)
MOV     4,D0             ; (2 bytes, 1 cycle)
BSET    D0,(A0)          ; (2 bytes, 5 cycles)
                   Total   (7 bytes, 7 cycles)
```

2-1 5 Bit Clear

Size optimization

【Contents】

Normally AND instructions are used to clear bits. For byte accesses the BCLR instruction can be used in addition to an AND instruction. If the bits to be cleared all fit within a byte, then a byte access will give smaller code than a word access.

(!) Use the BCLR instruction to prohibit reception of bus release requests and interrupts during read/modify/writes.

【Example】

```
① Word Access
MOV     (x'fc20'),D0      ; (3 bytes, 1 cycle)
AND     x'fffb',D0        ; (4 bytes, 2 cycles)
MOV     D0,(x'fc20')      ; (3 bytes, 1 cycle)
                   Total    (10 bytes, 4 cycles)

          ↓ (The example below gives smaller code size.)

MOV     x'fc20',A0        ; (3 bytes, 1 cycle)
MOV     (A0),D0           ; (1 byte, 1 cycle)
AND     x'fffb',D0        ; (4 bytes, 2 cycles)
MOV     D0,(A0)           ; (1 byte, 1 cycle)
                   Total    (9 bytes, 5 cycles)

② Using AND Instruction For Byte Access
MOVBU   (x'fc20'),D0      ; (3 bytes, 1 cycle)
AND     x'fb',D0          ; (3 bytes, 2 cycles)
MOVB    D0,(x'fc20')      ; (3 bytes, 1 cycle)
                   Total    (9 bytes, 4 cycles)

          ↓ (The example below gives smaller code size.)

MOV     x'fc20',A0        ; (3 bytes, 1 cycle)
MOVBU   (A0),D0           ; (1 bytes, 1 cycle)
AND     x'fb',D0          ; (3 bytes, 2 cycles)
MOVB    D0,(A0)           ; (1 bytes, 1 cycle)
                   Total    (8 bytes, 5 cycles)

③ Using BCLR Instruction For Byte Access
MOV     x'fc20',A0        ; (3 bytes, 1 cycle)
MOV     x'04',D0          ; (2 bytes, 1 cycle)
BCLR    D0,(A0)           ; (2 bytes, 5 cycles)
                   Total    (7 bytes, 7 cycles)
```

2-16 Bit Test

General information

【Contents】

Use the BTST instruction to test bits. If it does not matter whether or not the value in a register moved from memory is valid, then an AND instruction can also be used, but code size and processing speed will not change.

【Example】

```
① Register data must be valid.
MOV     (x'fc20'),D0        ; (3 bytes, 1 cycle)
BTST    x'5',D0             ; (3 bytes, 2 cycles)
                     Total  (6 bytes, 3 cycles)

② Register data may be invalid.
MOV     (x'fc20'),D0        ; (3 bytes, 1 cycle)
AND     x'5',D0             ; (3 bytes, 2 cycles)
                     Total  (6 bytes, 3 cycles)
```

2-17 Subtracting 1～128

Speed optimization   Size optimization

【Contents】

When subtracting values 1～128, an ADD instruction will result in faster, smaller code than a SUB instruction.

【Example】

```
SUB     x'7f',D0            ; (4 bytes, 2 cycles)
SUB     x'7f',A0            ; (4 bytes, 2 cycles)
                     Total  (8 bytes, 4 cycles)

        ↓ (The example below gives smaller, faster code.)

ADD     -x'7f',D0           ; (2 bytes, 1 cycle)
ADD     -x'7f',A0           ; (2 bytes, 1 cycle)
                     Total  (4 bytes, 2 cycles)
```

2-18 Sign Inversion

Speed optimization   Size optimization

【Contents】

Sign inversion is performed by a sequence of instructions. Example ② below needs two registers, but it results in faster, smaller code.

【Example】

```
① Use one data register.
NOT    D0              ; (2 bytes, 2 cycles)
ADD    1,D0            ; (2 bytes, 1 cycle)
                 Total  (4 bytes, 3 cycles)

② Use two data registers.
SUB    D1,D1           ; (1 byte, 1 cycle)
SUB    D0,D1           ; (1 byte, 1 cycle)
                 Total  (2 bytes, 2 cycles)
```

2-19 Logical Single-Bit Shift Left

Speed optimization   Size optimization

【Contents】

To multiply a register value by 2 or 4, or in other words to perform a logical shift left of one or two bits, use the ADD instruction.

【Example】

```
① 2x
AND    x'fffb',PSW     ; (4 bytes, 3 cycles)
ROL    D0              ; (2 bytes, 2 cycles)
                 Total  (6 bytes, 5 cycles)

         ↓

ADD    D0,D0           ; (1 byte, 1 cycle)
                 Total  (1 byte, 1 cycle)

② 4x
AND    x'fffb',PSW     ; (4 bytes, 3 cycles)
ROL    D0              ; (2 bytes, 2 cycles)
AND    x'fffb',PSW     ; (4 bytes, 3 cycles)
ROL    D0              ; (2 bytes, 2 cycles)
                 Total  (12 bytes, 10 cycles)

         ↓

ADD    D0,D0           ; (1 byte, 1 cycle)
ADD    D0,D0           ; (1 byte, 1 cycle)
                 Total  (2 bytes, 2 cycles)
```

2-20 Logical Multiple-Bit Shift

Speed optimization

【Contents】

A multiply instruction using a look-up table for the multiplier levels processing times for multibit shifts left or right to a fixed number of cycles, speeding up execution for all sizes greater than 3.

【Example】

① Left shift of any number of bits

```
;        D1<<=D2
Nlsl:    BRA     Nlsl1          ; (2 bytes, 2 cycles)
Nlsl0:   ADD     D1,D1          ; (1 byte, 1 cycle)
Nlsl1:   ADD     -1,D2          ; (2 bytes, 1 cycle)
         BNC     Nlsl0          ; (2 bytes, 2/1 cycle)
                              Total (7 bytes, 4+d2*4 cycles )
                                    Small ROM size
                              4~60 process cycles

Nlsl:    MOV     Ntbl,A1        ; (5 bytes, 3 cycles)
         ADD     D2,D2          ; (1 byte, 1 cycle)
         MOV     (D2,A1),D2     ; (2 bytes, 2 cycles)
         MULU    D2,D1          ; (2 bytes, 12 cycles)
                              Total (10+32 bytes, 18 cycles)
                              Large ROM size
                              Fixed number of process cycles
Ntbl:
         DW      x'0001'
         DW      x'0002'
         DW      x'0004'
          ∫
         DW      x'8000'
```

② Right shift of any number of bits

```
;        D1>>=D2
Nlsr:    BRA     Nlsr1          ; (2 bytes, 2 cycles)
Nlsr0:   LSR     D1             ; (2 bytes, 2 cycles)
Nlsr1:   ADD     -1,D2          ; (2 bytes, 1 cycle)
         BNC     Nlsr0          ; (2 bytes, 2/1 cycle)
                              Total (8 bytes, 4+d2*5 cycles)
                              Small ROM size
                              4~75 process cycles

Nlsr:    MOV     Ntbl+32,A1     ; (5 bytes, 3 cycles)
         ADD     D2,D2          ; (1 byte, 1 cycle)
         SUB     D2,A1          ; (2 bytes, 2 cycles)
         MOV     (A1),D2        ; (1 byte, 1 cycle)
         MULU    D2,D1          ; (2 bytes, 12 cycles)
         MOV     MDR,D1         ; (2 bytes, 2 cycles)
                              Total (13+32 bytes, 21 cycles)
                              Large ROM size
                              Fixed number of process cycles
Ntbl:
         DW      x'0001'      This code does not work properly
         DW      x'0002'      for an input value of 0 for D2.
         DW      x'0004'
          ∫
         DW      x'8000'
```

2-21 8-Bit Swap

Speed optimization   Size optimization

【Contents】

When performing an 8-bit swap (exchange upper 8 bits and lower 8 bits), sequence of instructions will result in smaller or faster code than a shift instruction.

【Example】

```
D0 swap
MOV     D0,D1                                   ; (1 byte, 1 cycle)
ADD     D0,D0      ; 1 bit shift left           ; (1 byte, 1 cycle)
ADD     D0,D0      ; 1 bit shift left           ; (1 byte, 1 cycle)
ADD     D0,D0      ; 1 bit shift left           ; (1 byte, 1 cycle)
ADD     D0,D0      ; 1 bit shift left           ; (1 byte, 1 cycle)
ADD     D0,D0      ; 1 bit shift left           ; (1 byte, 1 cycle)
ADD     D0,D0      ; 1 bit shift left           ; (1 byte, 1 cycle)
ADD     D0,D0      ; 1 bit shift left           ; (1 byte, 1 cycle)
ADD     D0,D0      ; 1 bit shift left           ; (1 byte, 1 cycle)
LSR     D1         ; 1 bit shift right          ; (2 bytes, 2 cycles)
LSR     D1         ; 1 bit shift right          ; (2 bytes, 2 cycles)
LSR     D1         ; 1 bit shift right          ; (2 bytes, 2 cycles)
LSR     D1         ; 1 bit shift right          ; (2 bytes, 2 cycles)
LSR     D1         ; 1 bit shift right          ; (2 bytes, 2 cycles)
LSR     D1         ; 1 bit shift right          ; (2 bytes, 2 cycles)
LSR     D1         ; 1 bit shift right          ; (2 bytes, 2 cycles)
LSR     D1         ; 1 bit shift right          ; (2 bytes, 2 cycles)
OR      D1,D0                                   ; (2 bytes, 2 cycles)
                                          Total   (27 bytes, 27 cycles)

            ↓ (Use of stack gives faster code.)

ADD     -2,A3                                   ; (2 bytes, 1 cycle)
MOV     D0,(A3)                                 ; (1 byte, 1 cycle)
MOVBU   (1,A3),D1                               ; (3 bytes, 2 cycles)
MOVB    D1,(A3)                                 ; (1 byte, 1 cycle)
MOVB    D0,(1,A3)                               ; (3 bytes, 2 cycles)
MOV     (A3),D0                                 ; (1 byte, 1 cycle)
ADD     2,A3                                    ; (2 bytes, 1 cycle)
                                          Total   (13 bytes, 9 cycles)

            ↓ (Use of multiply instruction gives smaller code.)

MOV     x'0100',D1                              ; (3 bytes, 1 cycle)
MULU    D1,D0                                   ; (2 bytes, 12 cycles)
MOV     MDR,D1                                  ; (2 bytes, 2 cycles)
ADD     D1,D0                                   ; (1 byte, 1 cycle)
                                          Total   (8 bytes, 16 cycles)
```

2-2 2 Decimal Conversion Of 4-Bit Data

General information

【Contents】

Below is an example of decimal conversion of 4-bit data.

【Example】

```
        AND     x'f',D0         ;Data before conversion is in
        CMP     10,D0           ;register D0
        BLS     label
        ADD     x'6',D0
label:                          ;Data after conversion is in register D0
```

2-2 3 Interrupt Disable/Enable

General information

【Contents】

The method for disabling and enabling interrupts is as follows. However, non-maskable interrupts cannot be disabled.

【Example】

```
① Disable interrupts (IE=0)
AND     x'f7ff',PSW

② Enable interrupts (IE=1)
OR      x'0800',PSW
```

2-2 4 PSW Flags Set/Clear

Speed optimization

Size optimization

【Contents】

Example ① shows the basic format for setting and clearing flags in the PSW, but other instructions can also be used as shown below.

【Example】

```
① Basic format
AND  imm,PSW                 ; (4 bytes, 3 cycles)
OR   imm,PSW                 ; (4 bytes, 3 cycles)

② Size and speed optimization
SUB  Dn,Dn  ; ZF set, NF,VF,CF clear,Dn clear    (1 byte, 1 cycle)
ADD  0,Dn   ; VF,CF clear                        (2 bytes, 1 cycle)
XOR  Dn,Dn  ; ZF set, NF,VF,CF clear,Dn clear    (2 bytes, 2cycles)
BTST 0,Dn   ; ZF set, NF,VF,CF clear             (2 bytes, 1 cycle)
CMP  0,Dn   ; VF,CF clear                        (2 bytes, 1 cycle)
AND  0,Dn   ; ZF set, NF,VF,CF clear,Dn clear    (2 bytes, 1 cycle)
AND  Dn,Dn  ; VF,CF clear                        (2 bytes, 2 cycles)
OR   0,Dn   ; VF,CF clear                        (2 bytes, 1 cycle)
OR   Dn,Dn  ; VF,CF clear                        (2 bytes, 2 cycles)
NOT  Dn     ; VF,CF clear,Dn changes             (2 bytes, 2 cycles)
```

2-2 5 Overlapping Interrupts

General information

【Contents】

The following example shows overlapping interrupts.

【Example】

```
irg     ADD     -30,A3
        MOV     A0,(A3)         ;Save registers to be used.
        MOV     A1,(4,A3)
        MOV     A2,(8,A3)
        MOVX    D0,(12,A3)
        MOVX    D1,(16,A3)
        MOVX    D2,(20,A3)
        MOVX    D3,(24,A3)
        MOV     MDR,D0
        MOV     D0,(28,A3)
        OR      x'0800',PSW     ;Enable overlapping interrupts.
                  :
            Interrupt process
                  :
        AND     x'f7ff',PSW     ;Disable overlapping interrupts.
        MOV     (28,A3),D0      ;Restore registers.
        MOV     D0,MDR
        MOVX    (24,A3),D3
        MOVX    (20,A3),D2
        MOVX    (16,A3),D1
        MOVX    (12,A3),D0
        MOVX    (8,A3),A2
        MOVX    (4,A3),A1
        MOV     (A3),A0
        ADD     30,A3
        RTI
```

3 Deleted Instructions

3-1 MOV (Di, An), Am, MOV Am, (Di, An)

The instruction above has been removed from the instruction set currently and it is not available to use. You are required to change the program in case of using the instruction.

■ Instruction replacement example

(Note: Flags are chnaged as a result of arithmetic operations.)

mov (Dn,An),Am →	add Dn,An	mov Am,(Dn,An) →	add Dn,An
	mov (An),Am		mov Am,(An)
	sub Dn,An		sub Dn,An

Appendix 4

MN102L SERIES INSTRUCTION SET

Instruction	Mnemonic	Operation	OP EX.	Flag								Code Size	Cycle	Machine Code	Page
				VX	CX	NX	ZX	VF	CF	NF	ZF				
MOV	MOV Dm,An	Dm→An	—	—	—	—	—	—	—	—	—	2	2	F2:30+Dm<<2+An	26
	MOV An,Dm	An→Dm	—	—	—	—	—	—	—	—	—	2	2	F2:F0+An<<2+Dm	26
	MOV Dn,Dm	Dn→Dm	—	—	—	—	—	—	—	—	—	1	1	80+Dn<<2+Dm *1	27
	MOV An,Am	An→Am	—	—	—	—	—	—	—	—	—	2	2	F2:70+An<<2+Am	27
	MOV PSW,Dn	PSW→Dn	0	—	—	—	—	—	—	—	—	2	2	F3:F0+Dn	28
	MOV Dn,PSW	Dn→PSW	—	●	●	●	●	●	●	●	●	2	3	F3:D0+Dn<<2	28
	MOV MDR,Dn	MDR→Dn	0	—	—	—	—	—	—	—	—	2	2	F3:E0+Dn	29
	MOV Dn,MDR	Dn→MDR	—	—	—	—	—	—	—	—	—	2	2	F3:C0+Dn<<2	29
	MOV (An),Dm	mem16(An)→Dm	S	—	—	—	—	—	—	—	—	1	1	20+An<<2+Dm	30
	MOV (d8,An),Dm	mem16(An+d8)→Dm	S	—	—	—	—	—	—	—	—	2	1	60+An<<2+Dm:d8	30
	MOV (d16,An),Dm	mem16(An+d16)→Dm	S	—	—	—	—	—	—	—	—	4	2	F7:C0+An<<2+Dm:d16-l:d16-h	31
	MOV (d24,An),Dm	mem16(An+d24)→Dm	S	—	—	—	—	—	—	—	—	5	3	F4:80+An<<2+Dm:d24-l:d24-m:d24-h	31
	MOV (Di,An),Dm	mem16(An+Di)→Dm	S	—	—	—	—	—	—	—	—	2	2	F1:40+Di<<4+An<<2+Dm	32
	MOV (abs16),Dn	mem16(abs16)→Dn	S	—	—	—	—	—	—	—	—	3	1	C8+Dn:abs16-l:abs16-h	32
	MOV (abs24),Dn	mem16(abs24)→Dn	S	—	—	—	—	—	—	—	—	5	3	F4:C0+Dn:abs24-l:abs24-m:abs24-h	33
	MOV (An),Am	mem24(An)→Am	—	—	—	—	—	—	—	—	—	2	2	70+An<<2+Am:00 *2	33
	MOV (d8,An),Am	mem24(An+d8)→Am	—	—	—	—	—	—	—	—	—	2	2	70+An<<2+Am:d8	34
	MOV (d16,An),Am	mem24(An+d16)→Am	—	—	—	—	—	—	—	—	—	4	3	F7:B0+An<<2+Am:d16-l:d16-h	34
	MOV (d24,An),Am	mem24(An+d24)→Am	—	—	—	—	—	—	—	—	—	5	4	F4:F0+An<<2+Am:d24-l:d24-m:d24-h	35
	MOV (abs16),An	mem24(abs16)→An	—	—	—	—	—	—	—	—	—	4	3	F7:30+An:abs16-l:abs16-h	36
	MOV (abs24),An	mem24(abs24)→An	—	—	—	—	—	—	—	—	—	5	4	F4:D0+An:abs24-l:abs24-m:abs24-h	36
	MOV Dm,(An)	Dm→mem16(An)	—	—	—	—	—	—	—	—	—	1	1	00+An<<2+Dm	37
	MOV Dm,(d8,An)	Dm→mem16(An+d8)	—	—	—	—	—	—	—	—	—	2	1	40+An<<2+Dm:d8	37
	MOV Dm,(d16,An)	Dm→mem16(An+d16)	—	—	—	—	—	—	—	—	—	4	2	F7:80+An<<2+Dm:d16-l:d16-h	38
	MOV Dm,(d24,An)	Dm→mem16(An+d24)	—	—	—	—	—	—	—	—	—	5	3	F4:00+An<<2+Dm:d24-l:d24-m:d24-h	38
	MOV Dm,(Di,An)	Dm→mem16(An+Di)	—	—	—	—	—	—	—	—	—	2	2	F1:C0+Di<<4+An<<2+Dm	39
	MOV Dn,(abs16)	Dn→mem16(abs16)	—	—	—	—	—	—	—	—	—	3	1	C0+Dn:abs16-l:abs16-h	39
	MOV Dn,(abs24)	Dn→mem16(abs24)	—	—	—	—	—	—	—	—	—	5	3	F4:40+Dn:abs24-l:abs24-m:abs24-h	40
	MOV Am,(An)	Am→mem24(An)	—	—	—	—	—	—	—	—	—	2	2	50+An<<2+Am:00 *3	40
	MOV Am,(d8,An)	Am→mem24(An+d8)	—	—	—	—	—	—	—	—	—	2	2	50+An<<2+Am:d8	41
	MOV Am,(d16,An)	Am→mem24(An+d16)	—	—	—	—	—	—	—	—	—	4	3	F7:A0+An<<2+Am:d16-l:d16-h	41
	MOV Am,(d24,An)	Am→mem24(An+d24)	—	—	—	—	—	—	—	—	—	5	4	F4:10+An<<2+Am:d24-l:d24-m:d24-h	42
	MOV An,(abs16)	An→mem24(abs16)	—	—	—	—	—	—	—	—	—	4	3	F7:20+An:abs16-l:abs16-h	43
	MOV An,(abs24)	An→mem24(abs24)	—	—	—	—	—	—	—	—	—	5	4	F4:50+An:abs24-l:abs24-m:abs24-h	43
	MOV imm8,Dn	imm8→Dn	S	—	—	—	—	—	—	—	—	2	1	80+Dn<<2+Dn:imm8	44
	MOV imm16,Dn	imm16→Dn	S	—	—	—	—	—	—	—	—	3	1	F8+Dn:imm16-l:imm16-h	44
	MOV imm24,Dn	imm24→Dn	—	—	—	—	—	—	—	—	—	5	3	F4:70+Dn:imm24-l:imm24-m:imm24-h	45
	MOV imm16,An	imm16→An	0	—	—	—	—	—	—	—	—	3	1	DC+An:imm16-l:imm16-h	45
	MOV imm24,An	imm24→An	—	—	—	—	—	—	—	—	—	5	3	F4:74+An:imm24-l:imm24-m:imm24-h	46
MOVX	MOVX (d8,An),Dm	mem24(An+d8)→Dm	—	—	—	—	—	—	—	—	—	3	3	F5:70+An<<2+Dm:d8	47
	MOVX (d16,An),Dm	mem24(An+d16)→Dm	—	—	—	—	—	—	—	—	—	4	3	F7:70+An<<2+Dm:d16-l:d16-h	47
	MOVX (d24,An),Dm	mem24(An+d24)→Dm	—	—	—	—	—	—	—	—	—	5	4	F4:B0+An<<2+Dm:d24-l:d24-m:d24-h	48
	MOVX Dm,(d8,An)	Dm→mem24(An+d8)	—	—	—	—	—	—	—	—	—	3	3	F5:50+An<<2+Dm:d8	48
	MOVX Dm,(d16,An)	Dm→mem24(An+d16)	—	—	—	—	—	—	—	—	—	4	3	F7:60+An<<2+Dm:d16-l:d16-h	49
	MOVX Dm,(d24,An)	Dm→mem24(An+d24)	—	—	—	—	—	—	—	—	—	5	4	F4:30+An<<2+Dm:d24-l:d24-m:d24-h	49
MOVB	MOVB (An),Dm	mem8(An)→Dm	S	—	—	—	—	—	—	—	—	2	2	30+An<<2+Dm:B8+Dm *4	50
	MOVB (d8,An),Dm	mem8(An+d8)→Dm	S	—	—	—	—	—	—	—	—	3	2	F5:20+An<<2+Dm:d8	50
	MOVB (d16,An),Dm	mem8(An+d16)→Dm	S	—	—	—	—	—	—	—	—	4	2	F7:D0+An<<2+Dm:d16-l:d16-h	51
	MOVB (d24,An),Dm	mem8(An+d24)→Dm	S	—	—	—	—	—	—	—	—	5	3	F4:A0+An<<2+Dm:d24-l:d24-m:d24-h	51
	MOVB (Di,An),Dm	mem8(An+Di)→Dm	S	—	—	—	—	—	—	—	—	2	2	F0:40+Di<<4+An<<2+Dm	52
	MOVB (abs16),Dn	mem8(abs16)→Dn	S	—	—	—	—	—	—	—	—	4	2	CC+Dn:abs16-l:abs16--h:B8+Dn *5	52
	MOVB (abs24),Dn	mem8(abs24)→Dn	S	—	—	—	—	—	—	—	—	5	3	F4:C4+Dn:abs24-l:abs24-m:abs24-h	53
	MOVB Dm,(An)	Dm→mem8(An)	—	—	—	—	—	—	—	—	—	1	1	10+Dm<<2+An	53
	MOVB Dm,(d8,An)	Dm→mem8(An+d8)	—	—	—	—	—	—	—	—	—	3	2	F5:10+An<<2+Dm:d8	54
	MOVB Dm,(d16,An)	Dm→mem8(An+d16)	—	—	—	—	—	—	—	—	—	4	2	F7:90+An<<2+Dm:d16-l:d16-h	54
	MOVB Dm,(d24,An)	Dm→mem8(An+d24)	—	—	—	—	—	—	—	—	—	5	3	F4:20+An<<2+Dm:d24-l:d24-m:d24-h	55
	MOVB Dm,(Di,An)	Dm→mem8(An+Di)	—	—	—	—	—	—	—	—	—	2	2	F0:C0+Di<<4+An<<2+Dm	55

Notes:
1* It is not possible to specify that Dn=Dm.
2* This instruction is supported by the assembler. For "MOV (d8,An),Am" the assembler will generate a bit pattern for d8=0.
3* This instruction is supported by the assembler. For "MOV Am,(d8,An)" the assembler will generate a bit pattern for d8=0.
4* This instruction is supported by the assembler. The assembler generates bit patterns for the two instructions "MOVBU (An),Dm" and "EXTXB Dm".
5* This instruction is supported by the assembler. The assembler generates bit patterns for the two instructions "MOVBU (abs16),Dn" and "EXTXB Dn".

Instruction	Mnemonic	Operation	OP EX.	Flag								Code Size	Cycle	Machine Code	Page
				VX	CX	NX	ZX	VF	CF	NF	ZF				
MOVB	MOVB Dn,(abs16)	Dn→mem8(abs16)	—	—	—	—	—	—	—	—	—	3	1	C4+Dn:abs16-l:abs16-h	56
	MOVB Dn,(abs24)	Dn→mem8(abs24)	—	—	—	—	—	—	—	—	—	5	3	F4:44+Dn:abs24-l:abs24-m:abs24-h	56
MOVBU	MOVBU (An),Dm	mem8(An)→Dm	0	—	—	—	—	—	—	—	—	1	1	30+An<<2+Dm	57
	MOVBU (d8,An),Dm	mem8(An+d8)→Dm	0	—	—	—	—	—	—	—	—	3	2	F5:30+An<<2+Dm:d8	57
	MOVBU (d16,An),Dm	mem8(An+d16)→Dm	0	—	—	—	—	—	—	—	—	4	2	F7:50+An<<2+Dm:d16-l:d16-h	58
	MOVBU (d24,An),Dm	mem8(An+d24)→Dm	0	—	—	—	—	—	—	—	—	5	3	F4:90+An<<2+Dm:d24-l:d24-m:d24-h	58
	MOVBU (Di,An),Dm	mem8(An+Di)→Dm	0	—	—	—	—	—	—	—	—	2	2	F0:80+Di<<4+An<<2+Dm	59
	MOVBU (abs16),Dn	mem8(abs16)→Dn	0	—	—	—	—	—	—	—	—	3	1	CC+Dn:abs16-l:abs16-h	59
	MOVBU (abs24),Dn	mem8(abs24)→Dn	0	—	—	—	—	—	—	—	—	5	3	F4:C8+Dn:abs24-l:abs24-m:abs24-h	60
EXT	EXT Dn	IF Dn.bp15=0 x'0000'→MDR IF Dn.bp15=1 x'FFFF'→MDR	S	—	—	—	—	—	—	—	—	2	3	F3:C1+Dn<<2 *6	61
EXTX	EXTX Dn	IF Dn.bp15=0 Dn&x'00FFFF'→Dn IF Dn.bp15=1 Dn|x'FF0000'→Dn	S	—	—	—	—	—	—	—	—	1	1	B0+Dn *7	62
EXTXU	EXTXU Dn	Dn&x'00FFFF'→Dn	0	—	—	—	—	—	—	—	—	1	1	B4+Dn *8	63
EXTXB	EXTXB Dn	IF Dn.bp7=0 Dn&x'0000FF'→Dn IF Dn.bp7=1 Dn|x'FFFF00'→Dn	S	—	—	—	—	—	—	—	—	1	1	B8+Dn *9	64
EXTXBU	EXTXBU Dn	Dn&x'0000FF'→Dn	0	—	—	—	—	—	—	—	—	1	1	BC+Dn *10	65
ADD	ADD Dn,Dm	Dm+Dn→Dm	—	●	●	●	●	●	●	●	●	1	1	90+Dn<<2+Dm	66
	ADD Dm,An	An+Dm→An	—	●	●	●	●	●	●	●	●	2	2	F2:00+Dm<<2+An	66
	ADD An,Dm	Dm+An→Dm	—	●	●	●	●	●	●	●	●	2	2	F2:C0+An<<2+Dm	67
	ADD An,Am	Am+An→Am	—	●	●	●	●	●	●	●	●	2	2	F2:40+An<<2+Am	67
	ADD imm8,Dn	Dn+imm8→Dn	S	●	●	●	●	●	●	●	●	2	1	D4+Dn:imm8	68
	ADD imm16,Dn	Dn+imm16→Dn	S	●	●	●	●	●	●	●	●	4	2	F7:18+Dn:imm16-l:imm16-h	68
	ADD imm24,Dn	Dn+imm24→Dn	—	●	●	●	●	●	●	●	●	5	3	F4:60+Dn:imm24-l:imm24-m:imm24-h	69
	ADD imm8,An	An+imm8→An	S	●	●	●	●	●	●	●	●	2	1	D0+An:imm8	69
	ADD imm16,An	An+imm16→An	S	●	●	●	●	●	●	●	●	4	2	F7:08+An:imm16-l:imm16-h	70
	ADD imm24,An	An+imm24→An	—	●	●	●	●	●	●	●	●	5	3	F4:64+An:imm24-l:imm24-m:imm24-h	70
ADDC	ADDC Dn,Dm	Dm+Dn+CF→Dm	—	●	●	●	●	●	●	●	●	2	2	F2:80+Dn<<2+Dm	71
ADDNF	ADDNF imm8,An	An+imm8→An	S	—	—	—	—	—	—	—	—	3	2	F5:0C+An:imm8 *11	72
SUB	SUB Dn,Dm	Dm-Dn→Dm	—	●	●	●	●	●	●	●	●	1	1	A0+Dn<<2+Dm	73
	SUB Dm,An	An-Dm→An	—	●	●	●	●	●	●	●	●	2	2	F2:10+Dm<<2+An	73
	SUB An,Dm	Dm-An→Dm	—	●	●	●	●	●	●	●	●	2	2	F2:D0+An<<2+Dm	74
	SUB An,Am	Am-An→Am	—	●	●	●	●	●	●	●	●	2	2	F2:50+An<<2+Am	74
	SUB imm16,Dn	Dn-imm16→Dn	S	●	●	●	●	●	●	●	●	4	2	F7:1C+Dn:imm16-l:imm16-h	75
	SUB imm24,Dn	Dn-imm24→Dn	—	●	●	●	●	●	●	●	●	5	3	F4:68+Dn:imm24-l:imm24-m:imm24-h	75
	SUB imm16,An	An-imm16→An	S	●	●	●	●	●	●	●	●	4	2	F7:0C+An:imm16-l:imm16-h	76
	SUB imm24,An	An-imm24→An	—	●	●	●	●	●	●	●	●	5	3	F4:6C+An:imm24-l:imm24-m:imm24-h	76
SUBC	SUBC Dn,Dm	Dm-Dn-CF→Dm	—	●	●	●	●	●	●	●	●	2	2	F2:90+Dn<<2+Dm	77
MUL	MUL Dn,Dm	Dm ✻ Dn→Dm (Dm ✻ Dn)>>16→MDR	—	?	?	?	?	0	?	●	●	2	12	F3:40+Dn<<2+Dm *12	78
MULU	MULU Dn,Dm	Dm ✻ Dn→Dm (Dm ✻ Dn)>>16→MDR	—	?	?	?	?	0	?	●	●	2	12	F3:50+Dn<<2+Dm *13	79
DIVU	DIVU Dn,Dm	(MDR<<16+Dm)/Dn→Dm ...MDR	—	?	?	0/?	●/?	0/1	?	●/?	●/?	2	13	F3:60+Dn<<2+Dm *14	80

Notes: 6* 32-bit sign extended word data
7* 24-bit sign extended word data
8* 24-bit zero extended word data
9* 24-bit sign extended byte data
10* 24-bit zero extended byte data
11* Addition without changing flag
12* 16x16 = 32 (signed)
13* 16x16 = 32 (unsigned)
14* 32÷16 = 16...16 (unsigned)

Instruction	Mnemonic	Operation	OP	Flag								Code	Cycle	Machine Code	Page
			EX.	VX	CX	NX	ZX	VF	CF	NF	ZF	Size			
CMP	CMP Dn,Dm	Dm-Dn	—	●	●	●	●	●	●	●	●	2	2	F3:90+Dn<<2+Dm	81
	CMP Dm,An	An-Dm	—	●	●	●	●	●	●	●	●	2	2	F2:20+Dm<<2+An	81
	CMP An,Dm	Dm-An	—	●	●	●	●	●	●	●	●	2	2	F2:E0+An<<2+Dm	82
	CMP An,Am	Am-An	—	●	●	●	●	●	●	●	●	2	2	F2:60+An<<2+Am	82
	CMP imm8,Dn	Dn-imm8	S	●	●	●	●	●	●	●	●	2	1	D8+Dn:imm8	83
	CMP imm16,Dn	Dn-imm16	S	●	●	●	●	●	●	●	●	4	2	F7:48+Dn:imm16-l:imm16-h	83
	CMP imm24,Dn	Dn-imm24	—	●	●	●	●	●	●	●	●	5	3	F4:78+Dn:imm24-l:imm24-m:imm24-h	84
	CMP imm16,An	An-imm16	0	●	●	●	●	●	●	●	●	3	1	EC+An:imm16-l:imm16-h	84
	CMP imm24,An	An-imm24	—	●	●	●	●	●	●	●	●	5	3	F4:7C+An:imm24-l:imm24-m:imm24-h	85
AND	AND Dn,Dm	Dm&(x'FF0000' |Dn)→Dm	—	—	—	—	—	0	0	●	●	2	2	F3:00+Dn<<2+Dm *15	86
	AND imm8,Dn	Dn&(x'FF0000' |imm8)→Dn	0	—	—	—	—	0	0	●	●	3	2	F5:00+Dn:imm8 *15	86
	AND imm16,Dn	Dn&(x'FF0000' |imm16)→Dn	—	—	—	—	—	0	0	●	●	4	2	F7:00+Dn:imm16-l:imm16-h *15	87
	AND imm16,PSW	PSW&imm16→PSW	—	●	●	●	●	●	●	●	●	4	3	F7:10:imm16-l:imm16-h *15	87
OR	OR Dn,Dm	Dm|(Dn&x'00FFFF')→Dm	—	—	—	—	—	0	0	●	●	2	2	F3:10+Dn<<2+Dm *15	88
	OR imm8,Dn	Dn|imm8→Dn	0	—	—	—	—	0	0	●	●	3	2	F5:08+Dn:imm8 *15	88
	OR imm16,Dn	Dn|imm16→Dn	—	—	—	—	—	0	0	●	●	4	2	F7:40+Dn:imm16-l:imm16-h *15	89
	OR imm16,PSW	PSW|imm16→PSW	—	●	●	●	●	●	●	●	●	4	3	F7:14:imm16-l:imm16-h *15	89
XOR	XOR Dn,Dm	Dm^(x'00FFFF'&Dn)→Dm	—	—	—	—	—	0	0	●	●	2	2	F3:20+Dn<<2+Dm *15	90
	XOR imm16,Dn	Dn^imm16→Dn	—	—	—	—	—	0	0	●	●	4	2	F7:4C+Dn:imm16-l:imm16-h *15	90
NOT	NOT Dn	Dn^x'00FFFF'→Dn	—	—	—	—	—	0	0	●	●	2	2	F3:E4+Dn *15	91
ASR	ASR Dn	Dn.lsb→CF Dn.bp→Dn.bp-1(bp15~1) Dn.bp15→Dn.bp15	—	—	—	—	—	0	●	●	●	2	2	F3:38+Dn *15	92
LSR	LSR Dn	Dn.lsb→CF Dn.bp→Dn.bp-1(bp15~1) 0→Dn.bp15	—	—	—	—	—	0	●	0	●	2	2	F3:3C+Dn *15	93
ROR	ROR Dn	Dn.lsb→temp Dn.bp→Dn.bp-1(bp15~1) CF→Dn.bp15 temp→CF	—	—	—	—	—	0	●	●	●	2	2	F3:34+Dn *15	94
ROL	ROL Dn	Dn.bp15→temp Dn.bp→Dn.bp+1(bp14~0) CF→Dn.lsb temp→CF	—	—	—	—	—	0	●	●	●	2	2	F3:30+Dn *15	95
BTST	BTST imm8,Dn	Dn&imm8 ... PSW	0	—	—	—	—	0	0	0	●	3	2	F5:04+Dn:imm8	96
	BTST imm16,Dn	Dn&imm16 ... PSW	0	—	—	—	—	0	0	●	●	4	2	F7:04+Dn:imm16-l:imm16-h	96
BSET	BSET Dm,(An)	mem8(An)&Dm ... PSW mem8(An)|Dm→mem8(An)	0	—	—	—	—	0	0	0	●	2	5	F0:20+An<<2+Dm *16	97
BCLR	BCLR Dm,(An)	mem8(An)&Dm ... PSW mem8(An)&(~Dm)→mem8(An)	0	—	—	—	—	0	0	0	●	2	5	F0:30+An<<2+Dm *16	98
Bcc	BEQ label	IF ZF=1 PC+2+d8(label)→PC IF ZF=0 PC+2→PC	—	—	—	—	—	—	—	—	—	2	2/1	E8:d8 *17	99
	BNE label	IF ZF=0 PC+2+d8(label)→PC IF ZF=1 PC+2→PC	—	—	—	—	—	—	—	—	—	2	2/1	E9:d8 *18	99
	BLT label	IF (VF^NF)=1 PC+2+d8(label)→PC IF (VF^NF)=0 PC+2→PC	—	—	—	—	—	—	—	—	—	2	2/1	E0:d8 *19	100

Notes: 15* 16-bit computation
16* Performed under the conditions of bus lock and disabled interrupts.
17* src=dest (lower 16 bits)
18* src≠dest (lower 16 bits)
19* src>dest (lower 16 bits, signed)

Instruction	Mnemonic	Operation	OP EX.	Flag VX	CX	NX	ZX	VF	CF	NF	ZF	Code Size	Cycle	Machine Code		Page
Bcc	BLE label	IF ((VF^NF)|ZF)=1 PC+2+d8(label)→PC IF ((VF^NF)|ZF)=0 PC+2→PC	—	—	—	—	—	—	—	—	—	2	2/1	E3:d8	*20	100
	BGE label	IF (VF^NF)=0 PC+2+d8(label)→PC IF (VF^NF)=1 PC+2→PC	—	—	—	—	—	—	—	—	—	2	2/1	E2:d8	*21	101
	BGT label	IF ((VF^NF)|ZF)=0 PC+2+d8(label)→PC IF ((VF^NF)|ZF)=1 PC+2→PC	—	—	—	—	—	—	—	—	—	2	2/1	E1:d8	*22	101
	BCS label	IF CF=1 PC+2+d8(label)→PC IF CF=0 PC+2→PC	—	—	—	—	—	—	—	—	—	2	2/1	E4:d8	*23	102
	BLS label	IF (CF|ZF)=1 PC+2+d8(label)→PC IF (CF|ZF)=0 PC+2→PC	—	—	—	—	—	—	—	—	—	2	2/1	E7:d8	*24	102
	BCC label	IF CF=0 PC+2+d8(label)→PC IF CF=1 PC+2→PC	—	—	—	—	—	—	—	—	—	2	2/1	E6:d8	*25	103
	BHI label	IF (CF|ZF)=0 PC+2+d8(label)→PC IF (CF|ZF)=1 PC+2→PC	—	—	—	—	—	—	—	—	—	2	2/1	E5:d8	*26	103
	BVC label	IF VF=0 PC+3+d8(label)→PC IF VF=1 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:FC:d8	*27	104
	BVS label	IF VF=1 PC+3+d8(label)→PC IF VF=0 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:FD:d8	*28	104
	BNC label	IF NF=0 PC+3+d8(label)→PC IF NF=1 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:FE:d8	*29	105
	BNS label	IF NF=1 PC+3+d8(label)→PC IF NF=0 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:FF:d8	*30	105
	BRA label	PC+2+d8(label)→PC	—	—	—	—	—	—	—	—	—	2	2	EA:d8		106
Bccx	BEQX label	IF ZX=1 PC+3+d8(label)→PC IF ZX=0 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:E8:d8	*31	107
	BNEX label	IF ZX=0 PC+3+d8(label)→PC IF ZX=1 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:E9:d8	*32	107

Notes: 20* src≥dest (lower 16 bits, signed)
21* src≤dest (lower 16 bits, signed)
22* src<dest (lower 16 bits, signed)
23* src>dest (lower 16 bits, unsigned)
24* src≥dest (lower 16 bits, unsigned)
25* src≤dest (lower 16 bits, unsigned)
26* src<dest (lower 16 bits, unsigned)
27* VF=0
28* VF=1
29* NF=0
30* NF=1
31* src=dest (24 bits)
32* src≠dest (24 bits)

Instruction	Mnemonic	Operation	OP EX.	Flag								Code Size	Cycle	Machine Code	Page
				VX	CX	NX	ZX	VF	CF	NF	ZF				
Bccx	BLTX label	IF (VX^NX)=1 PC+3+d8(label)→PC IF (VX^NX)=0 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:E0:d8 *33	108
	BLEX label	IF ((VX^NX)|ZX)=1 PC+3+d8(label)→PC IF ((VX^NX)|ZX)=0 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:E3:d8 *34	108
	BGEX label	IF (VX^NX)=0 PC+3+d8(label)→PC IF (VX^NX)=1 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:E2:d8 *35	109
	BGTX label	IF ((VX^NX)|ZX)=0 PC+3+d8(label)→PC IF ((VX^NX)|ZX)=1 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:E1:d8 *36	109
	BCSX label	IF CX=1 PC+3+d8(label)→PC IF CX=0 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:E4:d8 *37	110
	BLSX label	IF (CX|ZX)=1 PC+3+d8(label)→PC IF (CX|ZX)=0 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:E7:d8 *38	110
	BCCX label	IF CX=0 PC+3+d8(label)→PC IF CX=1 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:E6:d8 *39	111
	BHIX label	IF (CX|ZX)=0 PC+3+d8(label)→PC IF (CX|ZX)=1 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:E5:d8 *40	111
	BVCX label	IF VX=0 PC+3+d8(label)→PC IF VX=1 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:EC:d8 *41	112
	BVSX label	IF VX=1 PC+3+d8(label)→PC IF VX=0 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:ED:d8 *42	112
	BNCX label	IF NX=0 PC+3+d8(label)→PC IF NX=1 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:EE:d8 *43	113
	BNSX label	IF NX=1 PC+3+d8(label)→PC IF NX=0 PC+3→PC	—	—	—	—	—	—	—	—	—	3	3/2	F5:EF:d8 *44	113
JMP	JMP label16	PC+3+d16(label16)→PC	—	—	—	—	—	—	—	—	—	3	2	FC:d16-l:d16-h	114
	JMP label24	PC+5+d24(label24)→PC	—	—	—	—	—	—	—	—	—	5	4	F4:E0:d24-l:d24-m:d24-h	114
	JMP (An)	An→PC	—	—	—	—	—	—	—	—	—	2	3	F0:An<<2	115

Notes:
33* src>dest (24 bits, signed)
34* src≥dest (24 bits, signed)
35* src≤dest (24 bits, signed)
36* src<dest (24 bits, signed)
37* src>dest (24 bits, unsigned)
38* src≥dest (24 bits, unsigned)
39* src≤dest (24 bits, unsigned)
40* src<dest (24 bits, unsigned)
41* VX=0
42* VX=1
43* NX=0
44* NX=1

Instruction	Mnemonic	Operation	OP EX.	Flag								Code Size	Cycle	Machine Code	Page
				VX	CX	NX	ZX	VF	CF	NF	ZF				
JSR	JSR label16	A3-4→A3 PC+3→mem24(A3) PC+3+d16(label16)→PC	—	—	—	—	—	—	—	—	—	3	4	FD:d16-l:d16-h	116
	JSR label24	A3-4→A3 PC+5→mem24(A3) PC+5+d24(label24)→PC	—	—	—	—	—	—	—	—	—	5	5	F4:E1:d24-l:d24-m:d24-h	116
	JSR (An)	A3-4→A3 PC+2→mem24(A3) An→PC	—	—	—	—	—	—	—	—	—	2	5	F0:01+An<<2	117
NOP	NOP	PC+1→PC	—	—	—	—	—	—	—	—	—	1	1	F6	118
RTS	RTS	mem24(A3)→PC A3+4→A3	—	—	—	—	—	—	—	—	—	1	5	FE	119
RTI	RTI	mem16(A3)→PSW mem24(A3+2)→PC A3+6→A3	—	●	●	●	●	●	●	●	●	1	6	EB	120

Ver.2.0 (2001.02.01)

Reading the instruction set

■ Symbols used in tables

Dn, Dm, Di	Data register
An, Am	Address register
MDR, PSW, PC	Multiply/Divide Register, Processor Status Word, Program Counter
imm8, imm16, imm16-l, imm16-h imm24, imm24-l, imm24-m, imm24-h	Constant
d8, d16, d16-l, d16-h d24, d24-l, d24-m, d24-h	Displacement
abs16, abs16-l, abs16-h abs24, abs24-l, abs24-m, abs24-h	Absolute address
mem8 (An), mem8 (abs16), mem8 (abs24)	8-bit memory data which is determined by the address inside parentheses ( )
mem16 (An), mem16 (abs16), mem16 (abs24)	16-bit memory data which is determined by the address inside parentheses ( )
mem24 (Am), mem24 (abs16), mem24 (abs24)	24-bit memory data which is determined by the address inside parentheses ( )
.bp,.lsb, .msb	Bit specification
&, l , ^	Logical AND, logical OR, exclusive OR
~, <<	Bit inversion, bit shift
VX, CX, NX, ZX,	Extended overflow flag, carry flag, negative flag, zero flag (24-bit data)
VF, CF, NF, ZF	Overflow flag, carry flag, negative flag, zero flag (16-bit data)
temp	CPU internal temporary register
→ , ...	Substitution, reflects calculation results

■ OP EX. (Operand Extensions)

0	Zero-extension
S	Sign-extension
—	Not applicable

■ Flag

●	Changes
—	No change
0	Always 0
1	Always 1
?	Undefined

■ Code Size

Unit : bytes

■ Cycle

Minimum cycle count is shown.

Units : machine cycles

a/b : a cycles if branch taken
b cycles if branch not taken

■ Machine Code

":" indicates a delimiter between bytes.<<2 indicates a 2-bit shift.

Dn, Dm, Di, An, Am : Register numbers

D0	00	A0	00
D1	01	A1	01
D2	10	A2	10
D3	11	A3	11

■ Notes

- 16-bit or 24-bit access instruction must not access odd memory addresses.
- 8-bit displacements (d8) and 16-bit displacements (d16) are all sign-extended.

MN102L SERIES INSTRUCTION MAP

First byte Upper/Lower	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	MOV Dm, (An)															
1	MOVB Dm, (An)															
2	MOV (An), Dm															
3	MOVBU (An), Dm															
4	MOV Dm, (d8, An)															
5	MOV Am, (d8, An)															
6	MOV (d8, An), Dm															
7	MOV (d8, An), Am															
8	MOV Dn, Dm, (when src=dest,MOV imm8, Dn)															
9	ADD Dn, Dm															
A	SUB Dn, Dm															
B	EXTX Dn				EXTXU Dn				EXTXB Dn				EXTXBU Dn			
C	MOV Dn, (abs16)				MOVB Dn, (abs16)				MOV (abs16),Dn				MOVBU (abs16),Dn			
D	ADD imm8, An				ADD imm8, Dn				CMP imm8, Dn				MOV imm16, An			
E	BLT label	BGT label	BGE label	BLE label	BCS label	BHI label	BCC label	BLS label	BEQ label	BNE label	BRA label	RTI	CMP imm16, An			
F	Extended code (2 bytes)				Extended code (5 bytes)	Extended code (3 bytes)	NOP	Extended code (4 bytes)	MOV imm16, Dn				JMP label16	JSR label16	RTS	

2-byte instructions (Byte 1: F0)

Second byte Upper/Lower	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	JMP (A0)	JSR (A0)			JMP (A1)	JSR (A1)			JMP (A2)	JSR (A2)			JMP (A3)	JSR (A3)		
1																
2	BSET Dm, (An)															
3	BCLR Dm, (An)															
4–7	MOVB (Di, An), Dm															
8–B	MOVBU (Di, An), Dm															
C–F	MOVB Dm, (Di, An)															

2-byte instructions (Byte 1: F1)

Second byte Upper/Lower	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																
4	MOV (Di, An), Dm															
5																
6																
7																
8																
9																
A																
B																
C	MOV Dm, (Di, An)															
D																
E																
F																

2-byte instructions (Byte 1: F2)

Second byte Upper/Lower	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ADD Dm, An															
1	SUB Dm, An															
2	CMP Dm, An															
3	MOV Dm, An															
4	ADD An, Am															
5	SUB An, Am															
6	CMP An, Am															
7	MOV An, Am															
8	ADDC Dn, Dm															
9	SUBC Dn, Dm															
A																
B																
C	ADD An, Dm															
D	SUB An, Dm															
E	CMP An, Dm															
F	MOV An, Dm															

2-byte instructions (Byte 1: F3)

Second byte Upper/Lower	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	AND Dn, Dm															
1	OR Dn, Dm															
2	XOR Dn, Dm															
3	ROL Dn				ROR Dn				ASR Dn				LSR Dn			
4	MUL Dn, Dm															
5	MULU Dn, Dm															
6	DIVU Dn, Dm															
7																
8																
9	CMP Dn, Dm															
A																
B																
C	MOV D0, MDR	EXT D0			MOV D1, MDR	EXT D1			MOV D2, MDR	EXT D2			MOV D3, MDR	EXT D3		
D	MOV D0, PSW				MOV D1, PSW				MOV D2, PSW				MOV D3, PSW			
E	MOV MDR, Dn				NOT Dn											
F	MOV PSW, Dn															

5-byte instructions (Byte 1: F4)

Second byte Upper/Lower	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	MOV Dm, (d24, An)															
1	MOV Am, (d24, An)															
2	MOVB Dm, (d24, An)															
3	MOVX Dm, (d24, An)															
4	MOV Dn, (abs24)				MOVB Dn, (abs24)											
5	MOV An, (abs24)															
6	ADD imm24, Dn				ADD imm24, An				SUB imm24, Dn				SUB imm24, An			
7	MOV imm24, Dn				MOV imm24, An				CMP imm24, Dn				CMP imm24, An			
8	MOV (d24, An), Dm															
9	MOVBU (d24, An), Dm															
A	MOVB (d24, An), Dm															
B	MOVX (d24, An), Dm															
C	MOV (abs24), Dn				MOVB (abs24), Dn				MOVBU (abs24), Dn							
D	MOV (abs24), An															
E	JMP label24	JSR label24														
F	MOV (d24, An), Am															

3-byte instructions (Byte 1: F5)

<table>
<tr><th>Second byte Upper/Lower</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th><th>F</th></tr>
<tr><td>0</td><td colspan="4">AND imm8, Dn</td><td colspan="4">BTST imm8, Dn</td><td colspan="4">OR imm8, Dn</td><td colspan="4">ADDNF imm8, An</td></tr>
<tr><td>1</td><td colspan="16">MOVB Dm, (d8, An)</td></tr>
<tr><td>2</td><td colspan="16">MOVB (d8, An), Dm</td></tr>
<tr><td>3</td><td colspan="16">MOVBU (d8, An), Dm</td></tr>
<tr><td>4</td><td colspan="16"></td></tr>
<tr><td>5</td><td colspan="16">MOVX Dm, (d8, An)</td></tr>
<tr><td>6</td><td colspan="16"></td></tr>
<tr><td>7</td><td colspan="16">MOVX (d8, An), Dm</td></tr>
<tr><td>8</td><td colspan="16" rowspan="6"></td></tr>
<tr><td>9</td></tr>
<tr><td>A</td></tr>
<tr><td>B</td></tr>
<tr><td>C</td></tr>
<tr><td>D</td></tr>
<tr><td>E</td><td>BLTX label</td><td>BGTX label</td><td>BGEX label</td><td>BLEX label</td><td>BCSX label</td><td>BHIX label</td><td>BCCX label</td><td>BLSX label</td><td>BEQX label</td><td>BNEX label</td><td colspan="2"></td><td>BVCX label</td><td>BVSX label</td><td>BNCX label</td><td>BNSX label</td></tr>
<tr><td>F</td><td colspan="12"></td><td>BVC label</td><td>BVS label</td><td>BNC label</td><td>BNS label</td></tr>
</table>

4-byte instructions (Byte 1: F7)

<table>
<tr><th>Second byte Upper/Lower</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th><th>F</th></tr>
<tr><td>0</td><td colspan="4">AND imm16, Dn</td><td colspan="4">BTST imm16, Dn</td><td colspan="4">ADD imm16, An</td><td colspan="4">SUB imm16, An</td></tr>
<tr><td>1</td><td>AND imm16 PSW</td><td colspan="3"></td><td>OR imm16 PSW</td><td colspan="3"></td><td colspan="4">ADD imm16, Dn</td><td colspan="4">SUB imm16, Dn</td></tr>
<tr><td>2</td><td colspan="4">MOV An, (abs16)</td><td colspan="12" rowspan="2"></td></tr>
<tr><td>3</td><td colspan="4">MOV (abs16), An</td></tr>
<tr><td>4</td><td colspan="4">OR imm16, Dn</td><td colspan="4"></td><td colspan="4">CMP imm16, Dn</td><td colspan="4">XOR imm16, Dn</td></tr>
<tr><td>5</td><td colspan="16">MOVBU (d16, An), Dm</td></tr>
<tr><td>6</td><td colspan="16">MOVX Dm ,(d16, An)</td></tr>
<tr><td>7</td><td colspan="16">MOVX (d16, An), Dm</td></tr>
<tr><td>8</td><td colspan="16">MOV Dm, (d16, An)</td></tr>
<tr><td>9</td><td colspan="16">MOVB Dm, (d16, An)</td></tr>
<tr><td>A</td><td colspan="16">MOV Am, (d16, An)</td></tr>
<tr><td>B</td><td colspan="16">MOV (d16, An), Am</td></tr>
<tr><td>C</td><td colspan="16">MOV (d16, An), Dm</td></tr>
<tr><td>D</td><td colspan="16">MOVB (d16, An), Dm</td></tr>
<tr><td>E</td><td colspan="16" rowspan="2"></td></tr>
<tr><td>F</td></tr>
</table>

Index

5

Index

A

B

C

D

E

J

L

M

N

O

R

S

X

MN102L Series
Instruction Manual

April, 2001 3rd Edition

Issued by Matsushita Electric Industrial Co., Ltd.

Semiconductor Company, Matsushita Electric Industrial Co., Ltd.

Nagaokakyo, Kyoto, 617-8520 Japan
Tel: (075) 951-8151
http://www.panasonic.co.jp/semicon/

SALES OFFICES

■ U.S.A. SALES OFFICE

Panasonic Industrial Company [PIC]

- **New Jersey Office:**
  2 Panasonic Way, Secaucus, New Jersey 07094
  Tel: 201-392-6173
  Fax: 201-392-4652
- **Milpitas Office:**
  1600 McCandless Drive, Milpitas, California 95035
  Tel: 408-945-5630
  Fax: 408-946-9063
- **Chicago Office:**
  1707 N. Randall Road, Elgin, Illinois 60123-7847
  Tel: 847-468-5829
  Fax: 847-468-5725
- **Atlanta Office:**
  1225 Northbrook Parkway, Suite 1-151,
  Suwanee, Georgia 30174
  Tel: 770-338-6940
  Fax: 770-338-6849
- **San Diego Office:**
  9444 Balboa Avenue, Suite 185
  San Diego, California 92123
  Tel: 619-503-2940
  Fax: 619-715-5545

■ CANADA SALES OFFICE

Panasonic Canada Inc. [PCI]

5700 Ambler Drive Mississauga, Ontario, L4W 2T3
Tel: 905-624-5010
Fax: 905-624-9880

■ GERMANY SALES OFFICE

Panasonic Industrial Europe G.m.b.H. [PIEG]

- **Munich Office:**
  Hans-Pinsel-Strasse 2 85540 Haar
  Tel: 89-46159-156
  Fax: 89-46159-195

■ U.K. SALES OFFICE

Panasonic Industrial Europe Ltd. [PIEL]

- **Electric component Group:**
  Willoughby Road, Bracknell, Berkshire RG12 8FP
  Tel: 1344-85-3773
  Fax: 1344-85-3853

■ FRANCE SALES OFFICE

Panasonic Industrial Europe G.m.b.H. [PIEG]

- **Paris Office:**
  270, Avenue de President Wilson
  93218 La Plaine Saint-Denis Cedex
  Tel: 14946-4413
  Fax: 14946-0007

■ ITALY SALES OFFICE

Panasonic Industrial Europe G.m.b.H. [PIEG]

- **Milano Office:**
  Via Lucini N19, 20125 Milano
  Tel: 2678-8266
  Fax: 2668-8207

■ TAIWAN SALES OFFICE

Panasonic Industrial Sales Taiwan Co.,Ltd. [PIST]

- **Head Office:**
  6th Floor, Tai Ping & First Building No.550. Sec.4,
  Chung Hsiao E. Rd. Taipei 10516
  Tel: 2-2757-1900
  Fax: 2-2757-1906
- **Kaohsiung Office:**
  6th Floor, Hsien 1st Road Kaohsiung
  Tel: 7-223-5815
  Fax: 7-224-8362

■ HONG KONG SALES OFFICE

Panasonic Shun Hing Industrial Sales (Hong Kong) Co., Ltd. [PSI(HK)]

11/F, Great Eagle Centre, 23 Harbour Road,
Wanchai, Hong Kong.
Tel: 2529-7322
Fax: 2865-3697

■ SINGAPORE SALES OFFICE

Panasonic Semiconductor of South Asia [PSSA]

300 Beach Road # 16-01
The Concourse Singapore 199555
Tel: 390-3688
Fax: 390-3689

■ MALAYSIA SALES OFFICE

Panasonic Industrial Company (Malaysia) Sdn. Bhd.

- **Head Office: [PICM]**
  Tingkat 16B Menara PKNS PJ No.17,Jalan Yong
  Shook Lin 46050 Petaling Jaya Selangor Darul Ehsan
  Malaysia
  Tel: 03-7516606
  Fax: 03-7516666
- **Penang Office:**
  Suite 20-17,MWE PLAZA No.8,Lebuh Farquhar,10200
  Penang Malaysia
  Tel: 04-2625550
  Fax: 04-2619989
- **Johore Sales Office:**
  39-01 Jaran Sri Perkasa 2/1,Taman Tampoi
  Utama,Tampoi 81200 Johor Bahru,Johor Malaysia
  Tel: 07-241-3822
  Fax: 07-241-3996

■ CHINA SALES OFFICE

Panasonic SH Industrial Sales (Shenzhen) Co., Ltd. [PSI(SZ)]

7A-107, International Business & Exhibition Centre,
Futian Free Trade Zone, Shenzhen 518048
Tel: 755-359-8500
Fax: 755-359-8516

Panasonic Industrial (Shanghai) Co., Ltd. [PICS]

1F, Block A, Development Mansion, 51 Ri Jing Street,
Wai Gao Qiao Free Trade Zone, Shanghai 200137
Tel: 21-5866-6114
Fax: 21-5866-8000

■ THAILAND SALES OFFICE

Panasonic Industrial (Thailand) Ltd. [PICT]

252/133 Muang Thai-Phatra Complex Building,31st
Fl.Rachadaphisek Rd.,Huaykwang,Bangkok 10320
Tel: 02-6933407
Fax: 02-6933423

■ KOREA SALES OFFICE

Panasonic Industrial Korea Co., Ltd. [PIKL]

Hanil Group Bldg.11th Fl.,191 Hangangro 2ga,
Youngsans-ku,Seoul 140-702,Korea
Tel: 82-2-795-9600
Fax: 82-2-795-1542

■ PHILIPPINES SALES OFFICE

National Panasonic Sales Philippines [NPP]

102 Laguna Boulevard Laguna Technopark Sta.
Rosa. Laguna 4026 Philippines
Tel: 02-520-3150
Fax: 02-843-2778

130301

Printed in JAPAN